IAS Executive Facilities Reference Manual

Order Number: AA–H005B–TC

This manual describes the facilities available through the IAS Executive.

Operating System and Version: IAS Version 3.4

May 1990

The postpaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist in preparing future documentation.

This document was prepared using VAX DOCUMENT, Version 1.2

Contents

FIGURES

TABLES

Preface

Purpose of the Manual

The *IAS Executive Facilities Reference Manual* describes the facilities available to you through the IAS Executive.

You should have a basic knowledge of MACRO-11 and/or FORTRAN. Further, this manual assumes a knowledge of the information covered in the *IAS PDS User's Guide* or the *IAS MCR User's Guide*, as well as the PDP-11 processor handbook relevant to your processor type.

Document Structure

- Chapter 1 outlines the design philosophy of the IAS Executive by summarizing the PDP-11 hardware environment and describing how the system operates within the confines of the hardware architecture.
- Chapter 2 describes the basic services provided by the Executive, including system directives, events and event flags, system trapping mechanisms, and techniques for intertask communication.
- Chapter 3 describes in detail the most crucial components of the system data structures, including linked lists and fixed tables, node accounting, and the most significant fields of the structures themselves.
- Chapter 4 describes the three types of IAS partitions, the scheduling mechanism, the checkpointing and swapping capabilities of the sytem, the use of fixed tasks, and memory allocation.
- Chapter 5 describes the characteristics of shareable global areas (SGAs).
- Chapter 6 describes the input/output facilities available in the system by explaining logical units, queue I/O system directives, and spooling.
- Appendix A contains the formats of the system lists and tables that are described in detail in Chapter 3.
- Appendix B contains a listing of QIOMAC.LST.

Associated Documents

The following books are prerequisite sources of information for readers of this manual:

- *IAS PDS User's Guide*
- *IAS MCR User's Guide*
- PDP-11 processor handbook (relevant to your processor type)

Other documents related to the contents of this manual are described briefly in the *IAS Master Index and Documentation Directory*, which defines the intended audience of each manual in the IAS document set and provides a brief summary of the contents of each manual.

1 Executive Design Considerations

This chapter considers the design of the Executive by summarizing the PDP-11 hardware environment and describing how the system operates within the context of the hardware architecture.

The information is presented in three parts:

1 An overview of the hardware memory management facilities, describing the processor status word and the PDP-11 virtual and physical addressing capabilities (see Section 1.1).

2 A description of memory mapping for IAS, introducing the concepts of Active Page Registers (APRs) and the shared use of processor registers (see Section 1.2).

3 A brief summary of the communication between the Executive and tasks (see Section 1.3).

1.1 Hardware Memory Management

This section describes the importance of the fields within the Processor Status word, and the virtual and physical addressing capabilities of the PDP-11. Read the PDP-11 processor handbook relevant to your processor-type for a full description of the Memory Management hardware.

1.1.1 Processor Status Word

The Processor Status word (PS), located at UNIBUS address 177776, contains information regarding the current state of the processor. This information includes the following parameters:

1 The current processor priority.

2 The current and previous modes of operation.

3 The condition codes describing the results of the previous instructions.

4 A hardware facility for program debugging.

Figure 1–1 illustrates the layout of the PS.

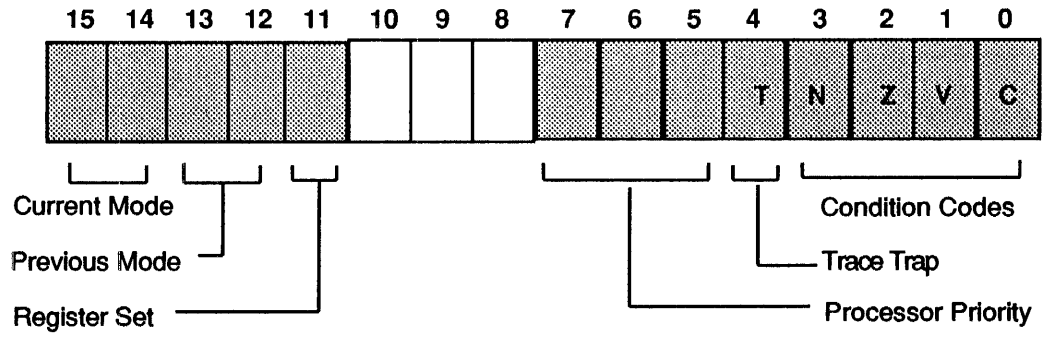
Current Mode (Bits 14-15)

This field determines the current processor mode (00 for Kernel Mode and 11 for User Mode), and is used to select a set of mapping registers for memory relocation, and a stack pointer (SP).

Previous Mode (Bits 12-13)

Whenever the PS is changed as a result of an interrupt or trap-type instruction, this field shows the previous operating mode (that is, bits 14-15 of the old PS are moved into bits 12-13 of the new PS).

Figure 1–1 Layout of the Processor Status Word



General Register Set (Bit 11)

This bit determines which general register is currently in use. On some PDP-11 processors, the hardware provides two sets of general purpose registers. On other supported processors, only one register set is provided. IAS uses only one set of registers and will use Register Set One if two sets are available.

Priority (Bits 5-7)

The processor operates at any of eight priority levels (0-7), determined by the value contained in bits 5-7. The levels 4-7 inclusive are associated with peripheral devices and are used for inhibiting device interrupts. Levels 0-3 inclusive are available for use by the system software and IAS uses them to indicate whether a user task or the Executive is running.

Trace Trap (Bit 4)

If this bit is set, a trap through location 14 will occur at the completion of each instruction. The trace trap is used for debugging aids to trace the execution of a program. See the appropriate PDP-11 Processor Handbook for details.

Condition Codes (Bits 0-3)

The condition codes contain the following information on the result of previous processor operations:

1 Carry bit (C) is set if the previous operation caused a carry out of its most significant bit.

2 Overflow bit (V) is set if the previous operation resulted in an arithmetical overflow.

3 Zero bit (Z) is set if the result of the previous operation was zero.

4 Negative bit (N) is set if the result of the previous operation was negative.

Not all instructions affect all condition codes. See the appropriate PDP-11 Processor Handbook for a complete description of the use of condition codes.

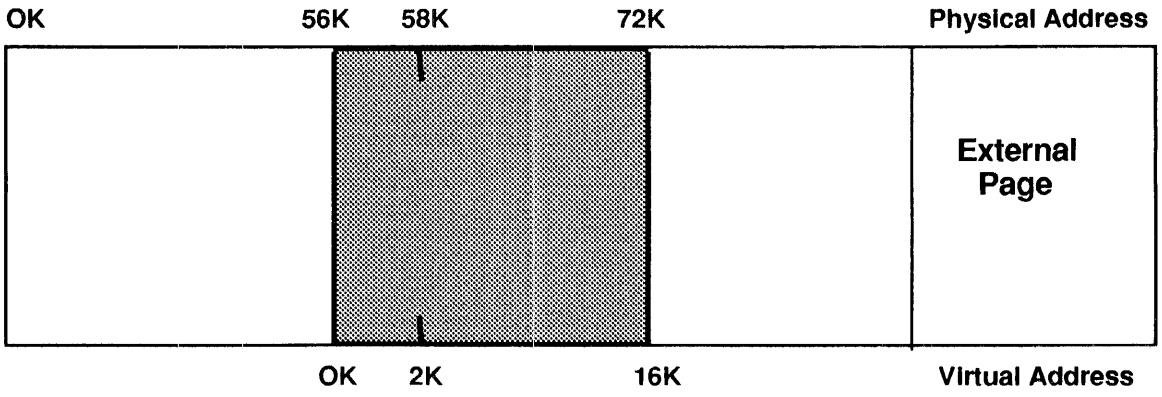
To change the Processor Status word explicitly, have an executing program reference it directly, just as the program would reference any other memory location. The PS can also be changed implicitly as a result of an interrupt, a trap, or the execution of a trap-type instruction. In this case, the new PS contents are taken from a predefined memory location (interrupt or trap vector) and used to set the whole PS. The old PS contents are saved on the current mode stack, where the current mode is determined by the new PS. When the PS is changed in this way, the previous mode field (bits 12-13) is set to show the processor mode before the interrupt or trap occurred. The value contained in the current mode (bits 14-15) of the old PS is moved into bits 12-13 of the new PS, regardless of any other setting indicated by the new PS in the interrupt vector.

1.1.2 Virtual and Physical Addressing

The PDP-11 uses 16-bit byte addressing and is thus able to directly address up to 64K bytes, or 32K words. IAS supports more than 32K words of physical memory and therefore requires additional hardware to gain access to this memory. When a program is executing, using 16-bit addresses to reference memory, the hardware must intervene to convert the 16-bit addresses into physical memory addresses. By convention, IAS uses the term virtual address to apply to the 16-bit address formed by a task and the term physical address to apply to the real address in the system's memory. The physical address length can be either 22-bits or 18-bits, depending on the processor type.

The hardware used to convert a program's virtual addresses into physical addresses is implemented by a set of eight registers known as memory mapping registers. Figure 1–2 illustrates an example of a task's physical and virtual addresses.

Figure 1–2 Correlation between Physical & Virtual Addresses



In this example, a 16K task (A) occupies memory between physical addresses 56K to 72K. When task A, for example, references its virtual address 2K, the memory mapping hardware must convert this 2K value and relocate it to the physical address of 58K.

The eight memory mapping registers are called Active Page Registers (APRs). Each APR is capable of relocating from 32 words to 4K words of memory. Therefore, a maximum of 32K words can be relocated by the set of eight APRs and consequently a task can have all of its 32K words of virtual address relocated by the memory mapping hardware.

Each APR consists of a pair of 16-bit registers, a Page Descriptor Register (PDR) that contains the page length and access rights, and a Page Address Register (PAR) that specifies where the page begins in memory.

The APR that performs the memory relocation is selected by the three most significant bits of the virtual address, as shown in Table 1–1.

Table 1–1 Virtual Address Ranges

000000—017776	0
020000—037776	1
040000—057776	2
060000—077776	3
100000—117776	4

Figure 1–2 illustrated an example of task A with a virtual address of 2K. This address (10000 octal) would be relocated by APR0; thus, PAR0 would point to the beginning of page 0 at physical address 56K.

Page lengths and physical memory are allocated in units of 32 words. A page length is specified as from 1 to 128 blocks of 32-words of memory. Physical memory is allocated in blocks of 32 words, starting at 32-word boundaries. A PAR is 16 bits in length and describes the start address of a 32-word block. Therefore, the 16-bit PAR can specify a 22-bit physical memory address.

The 32-word unit of allocation represents a compromise between two aims.

1 Large physical address space (which is increased by increasing the unit of allocation), and

2 Minimum wasted space at the end of each page.

The Memory Management hardware enables access to a page to be set to read/write or read-only, or allows access to the page to be denied altogether. When an APR is used to relocate a virtual address, the access rights are checked against the attempted access before the physical memory location is referenced.

APRs can be set up to map over any of the physical address space, that includes memory and the External Page. In a system running in 18-bit addressing mode, the External Page is located at physical addresses 124K to 128K. In order to access the External Page it is necessary to set up an APR that maps onto the 124K to 128K area. Because APRs are themselves registers on the External Page, the hardware must provide a mechanism for initially setting the APRs. When a system is bootstrapped, a hardware reset (hardware bootstrap) or software reset instruction (software bootstrap) is performed, which causes the Memory Management hardware to be disabled.

When memory management is disabled, virtual and physical addresses between 0K and 28K correspond directly, and virtual addresses between 28K and 32K are mapped onto the External Page. Therefore, when a system is first bootstrapped, it has access to the External Page and can set up the APRs so that the External Page can still be accessed after memory management has been enabled.

Two sets of APRs are used by IAS.

1 One set is used when the processor is in Kernel Mode.

2 Another set is used when the processor is in User Mode.

APRs can be set to map over any section of physical address space and it is possible for Kernel and User APRs to relocate to the same physical addresses. Alternatively, different virtual addresses can be relocated to the same physical address.

Before proceeding to the next section, you should understand the following points:

- The settings of the Processor Status word (PS) fields are of crucial importance to the operation of the processor.
- The setting of the current mode in the PS will determine which APR set is used and therefore where the processor will fetch instructions from physical memory.
- Any operation which can change the contents of the current mode bits in the PS can cause the processor to start fetching instructions from another address space in another part of physical memory.
- A new PS is loaded when an interrupt or trap occurs or a trap-type instruction is executed. Therefore, an interrupt or trap can cause a switch of processor modes.
- The PS can be changed by any executing program which has access to the External Page.
- A program with access to the External Page can change the memory management registers and can dynamically remap itself.
- User and Kernel APRs can map over the same physical memory.

1.2 Memory Mapping for IAS

The Memory Management hardware and memory mapping facilities have been described in Section 1.1 as they are available on the PDP-11. Any operating system can use these facilities in any chosen fashion. This section describes how IAS uses the memory management and memory mapping facilities.

IAS has five types of segments that must be mapped and relocated by the memory management hardware:

The Executive

The System Communication Area (SCOM)

The External Page

Tasks

Shareable Global Areas (SGAs) and dynamic regions.

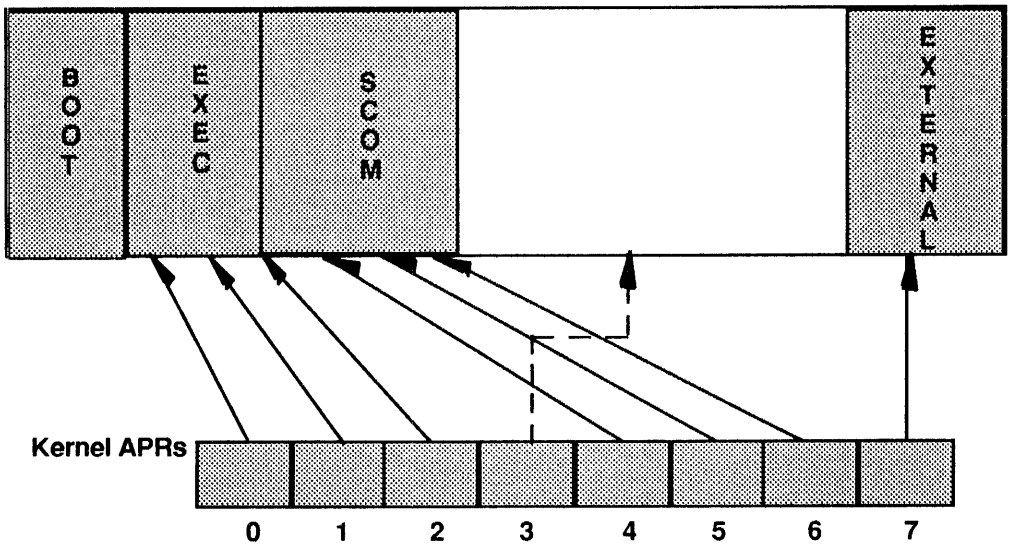
IAS requires that a virtual address area be relocated into one area of contiguous physical memory locations. However, when a task uses a shareable global area, some of that task's available APRs map over the SGA. Because a task area and an SGA area are separate segments, they can each be positioned anywhere in physical memory and therefore do not have to be adjacent in memory.

1.2.1 Active Page Registers (APRs)

As described in Section 1.1, the eight memory-mapping registers are called Active Page Registers (APRs). Each APR is capable of relocating from 32 words to 4K words of memory.

A set of Kernel APRs and User APRs are provided in the memory management unit. The current mode of the PS determines which set of APRs is used. Because all the processor facilities are available to programs executing in Kernel mode, the IAS Executive uses the Kernel APR set. Figure 1–3 illustrates how the Executive maps onto the eight Kernel APRs.

Figure 1–3 Kernel APR Mapping



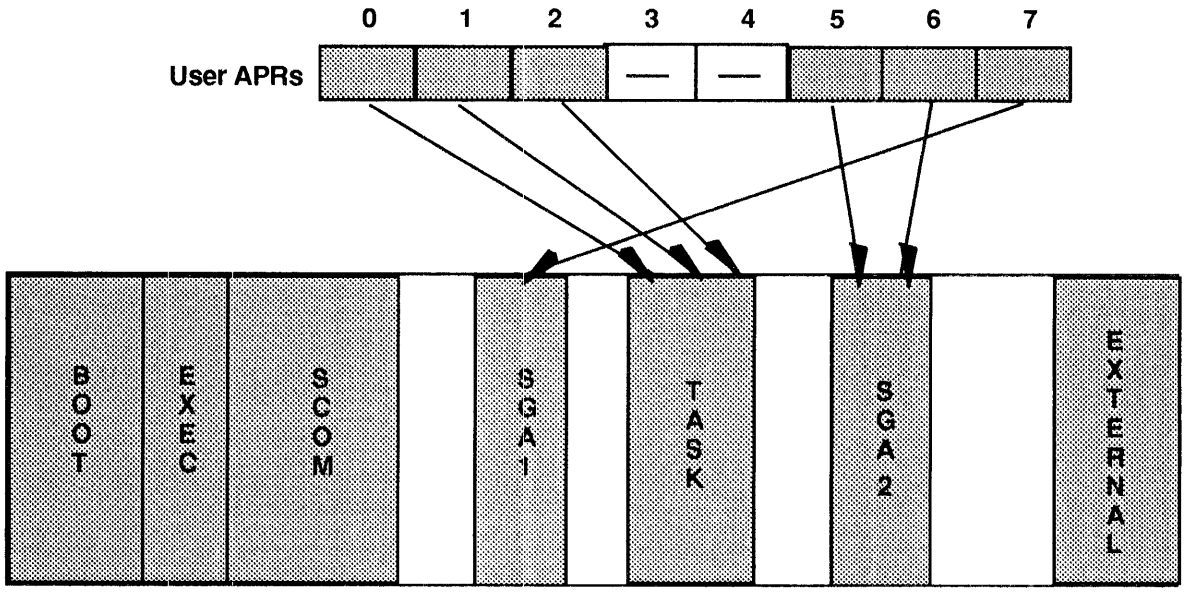
APR 0 Executive
APR 1 Executive
APR 2 Executive
APR 3 Changed dynamically by the Executive
APR 4 SCOM (not used if SCOM <8K)
APR 5 SCOM (not used if SCOM <4K)
APR 6 SCOM
APR 7 External Page

The Executive and SCOM normally occupy adjacent areas of physical memory, but do not have adjacent virtual addresses. The Executive needs access to all of physical memory and dynamically sets APR3 to map onto any part of memory.

User programs are executed in User mode and thus are mapped with the User APRs. This ensures that a program running in User mode is prevented from executing certain instructions that could cause the system to become corrupted.

Figure 1–4 illustrates a typical mapping of a user program. The example shows a task, 12K words in size, that maps onto two shareable global areas of 4K and 8K. Therefore, only 24K of the possible 32K address space is used, leaving two of the APRs (APR3 and APR4) free.

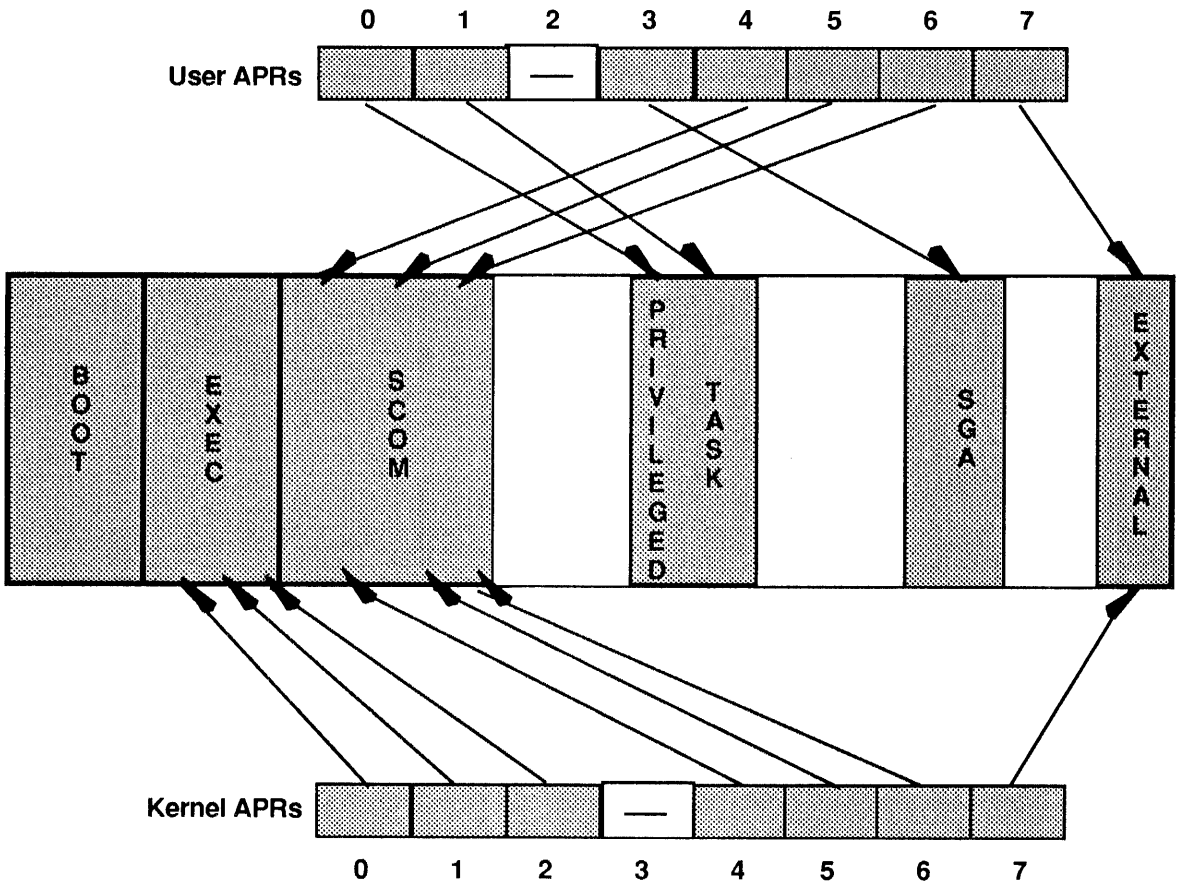
Figure 1–4 User APR Mapping



IAS provides the facility to run executive privileged tasks. These tasks run in User mode but are allowed access to SCOM and the External Page. SCOM is mapped using APRs 4, 5 and 6 and the External Page using APR7. Because these APRs are the same as those used by the Executive in the Kernel APR set, privileged tasks and the Executive can refer to locations in SCOM and the External Page using the same virtual address. Because four of the APRs are used, the maximum size of a privileged user task is 16K words. See Figure 1–5 for an example of the Executive and privileged user task mappings.

Parts of the IAS operating system software are executive privileged tasks. Typically these tasks perform operations on the data structures that do not need to be "instantaneous." An example is the MCR function task INSTALL, used to create an entry in the System Task Directory (STD). The use of privileged tasks in this way considerably reduces the size of the resident Executive. See Chapter 2, Section 2.5 for an overview of the system data structures. See Chapter 3 for a detailed description of the most significant fields within the data structures.

Figure 1–5 Mapping of the Executive and Privileged User Task



1.2.2 Shared Use of Processor Registers

It is important that the appropriate PDP-11 Processor Handbook and, in particular, the use of stack pointers (SP) and the program counter (PC) be understood before reading this section.

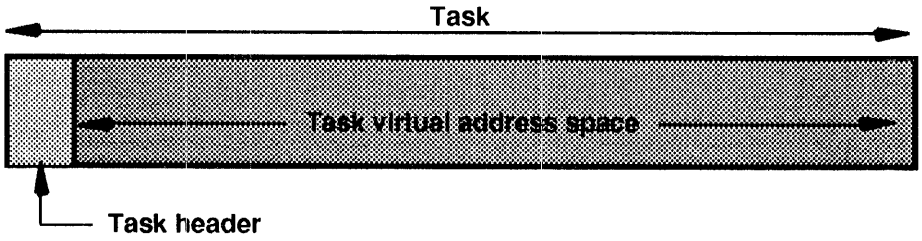
For a task to execute, it must have exclusive use of the following:

- The general registers, including the program counter (PC)
- The User APRs

Optionally, the floating point registers can be used.

Because IAS allows more than one task to be active, each active task must share the use of the processor registers. However, only one of the active tasks can be serviced at any given point in time and so the processor registers are set up for the use of one task at a time. When the Executive switches execution of tasks, the register contents for the currently active task must be saved and the register contents for the task which is about to start executing must be restored. Such saving and restoring of register contents by the Executive is called context switching. The register contents are stored in the task header (see Figure 1–6).

Figure 1–6 Task Header and Virtual Address Space



The task image consists of the task as seen by the user plus the task header. The task header is not part of the task's virtual address space and cannot be accessed by a normal user task. Because a privileged task has access to the User APRs it can reset an APR to map over any part of physical memory and can therefore access its own task header. The task header immediately precedes the task in physical memory and is resident at all times when the task is in core.

In addition to a context save area, the task header contains task data and parameters for controlling the execution of the task (see the *IAS Task Builder Reference Manual* for details).

1.3 Executive-Task Communication

This section describes the following concepts:

1 How a task uses the Emulator Trap (EMT) instruction to request the Executive to perform operations on its behalf.

2 How the Executive uses MFPI and MTPI instructions to communicate with the task.

1.3.1 EMT Instruction

IAS uses the Memory Management Hardware to provide an effective means of communication between the Executive and a user task. For instance, consider the case of an executing user task issuing a request to the Executive to perform an indicated operation (via a system directive).

The Kernel APRs are set up mapping the Executive, SCOM and the External Page; the User APRs are set up by mapping the executing user task. The task's request is issued in the form of a system directive, which is initiated by an Emulator Trap (EMT) instruction. Because an EMT is a trap-type instruction, a new PS and PC are taken from the EMT's trap vector at Kernel locations 30 and 32. The EMT vector's new PS is set up to cause a switch to Kernel mode. Therefore, the next instruction to be executed (at the new PC) will be taken from Kernel virtual address space. This will be part of the Executive's EMT servicing routine.

The Executive now processes the directive by obtaining the directive parameters from the user task and finally returning the user task with an indication of whether the directive was successful. The indication is returned in a location in the user task called the Directive Status Word (DSW).

1.3.2 MFPI and MTPI Instructions

When the processor needs to access data held in a task's virtual address space, the following instructions are used:

- Move From Previous Instruction Space (MFPI)
- Move To Previous Instruction Space (MTPI).

In the example of an EMT instruction (Section 1.3.1 above), when the task executed the EMT, the processor was running in User mode. The new PS causes the processor to be switched to Kernel mode. Therefore, the previous mode field in the new PS will be User mode and the current mode field will be Kernel (see Figure 1–1). The contents of the Kernel and User APRs have not been altered and will be mapping the Executive and user task as before.

The MFPI instruction enables the Executive to obtain information from the user task, using the fact that the User APRs are still mapped over the task. The MTPI instruction enables information such as the Directive Status Word to be returned to the task. MFPI and MTPI instructions make use of the APRs to perform memory relocation.

To simplify the servicing of a directive, the Executive always completes a request from a system directive before considering another task for execution. Furthermore, because the User APRs are already set up to allow the return of information, it is more efficient to restrict the Executive to servicing one directive request at a time.

Whenever memory relocation is performed, it is possible to form virtual addresses which are not mapped onto physical memory. If such a situation occurs, a segment fault trap will be performed by the processor. However, the IAS Executive is written assuming that segment faults should not occur within itself (that is, it is assumed that the Kernel APRs are set up correctly and that invalid addresses will not be used). User tasks that generate a segment fault cause a trap into the Executive and are either aborted or given a Synchronous System Trap (SST) by the Executive. See Section 2.3 for a description of SSTs.

Although user tasks sometimes operate inconsistently with respect to APR mapping, the Executive is written to handle this. However, when the Executive executes MFPI or MTPI instructions, it maps through the User APRs user-supplied addresses that might be inconsistent. In these situations, the Executive would receive segment faults and therefore the Executive has been written to recover from such faults and indicate an error condition to the issuing task.

2 Executive Services

This chapter describes the basic design elements of the IAS Executive and the various services available to the user. The information is presented as follows:

1 A brief description of system directives (see Section 2.1.

2 An introduction to *significant events* and event flags with examples of the use of event flags (see Sections 2.2 and 2.3).

3 A detailed description of asynchronous and synchronous system traps and examples of service routines which can be used to deal with these conditions (see Section 2.4).

4 Descriptions of the techniques for intertask communication and data transfer via event flags, dynamic regions, shareable global areas, SEND/RECEIVE data blocks and shared data files (see Section 2.5).

2.1 System Directives

The IAS Executive performs certain services upon request by a task, for example, task synchronization, intertask communication and input/output device transfers (in conjunction with the device handlers). These services are called system directives and are fully described in the *IAS System Directives Reference Manual*. As outlined in Section 1.3, the task uses an EMT and the user stack to pass parameters to the Executive that describe the directive. The Executive returns information to the task in a variety of ways, depending on the particular directive.

2.2 Significant Events

A *significant event* is a change in system status that causes the Executive to reevaluate which active tasks are eligible to run. A significant event is usually caused (either directly or indirectly) by a system directive issued from within a task. Examples of significant events include the following:

- Completion of an I/O request.
- A task exit.
- The occurrence of a situation declared explicitly by a task. For example, the execution of a SEND DATA (VSDA$/SDAT$), ALTER PRIORITY (ALTP$), RECEIVE BY REFERENCE (RREF$) or a DECLARE SIGNIFICANT EVENT (DECL$) system directive.
- The execution of an illegal instruction.
- The operation of the IAS scheduler.

2.3 Event Flags

Event flags are one means by which tasks can synchronize internally with themselves, with one another, and with operations performed on their behalf in the system. (Tasks also use Asynchronous System Traps (ASTs) to recognize specific events; see Section 2.4.3). When a task requests a system operation (such as an I/O transfer), the task can associate an event flag with the completion of the operation. When the event occurs, the Executive sets the specified flag. Section 2.3.1 describes in several examples how tasks can use event flags to coordinate task execution.

Sixty-four event flags are available to enable tasks to distinguish one event from another. Each event flag has a corresponding unique Event Flag Number (EFN), in the range 1-64. The first 32 (1-32) flags are unique to each task and are called local event flags. Each task has its own set of local event flags, and changes to one task's local flags have no effect on any other task's local flags. The second 32 flags (33-64) are common to all tasks and are called common or global event flags. The common flags are shared between all tasks and may therefore be used for intertask communication. The last eight flags in each group, local flags 25-32 and common flags 57-64, are reserved for use by the system and should not be explicitly referenced by a task.

The setting, clearing, and testing of all flags can be performed by using SET EVENT FLAG (SETF$), CLEAR EVENT FLAG (CLEF$), READ EVENT FLAG (RDEF$), and READ ALL FLAGS (RDAF$) directives.

The user must take great care when setting or clearing event flags, especially common flags. Erroneous or multiple setting and clearing of event flags can result in obscure software faults. A typical application program can be written without explicitly accessing or modifying event flags, since many of the directives can implicitly perform these functions. The SEND DATA (VSDA$/SDAT$), SEND BY REFERENCE (SREF$), MARK TIME (MRKT$) and the I/O operations directives can all implicitly alter an event flag. The implicit handling of event flags substantially reduces errors caused by multiple setting and clearing of event flags.

If the Executive rejects a directive, it usually does not clear or set any specified event flag. Thus, the task may wait forever if it indiscriminately executes a WAITFOR directive corresponding to a previously issued MARK TIME or QUEUE I/O directive that the Executive has rejected. Care should always be taken to ensure that a directive has completed successfully.

2.3.1 Using Event Flags

Examples 1 and 2 below illustrate the use of common event flags (33-64) to synchronize task execution. Examples 3 and 4 illustrate the use of local flags (1-32).

Example 1

Task B clears common event flag 35 and then blocks itself by issuing a WAITFOR directive that specifies common event flag 35.

Subsequently another task, Task A, specifies event flag 35 in a SET EVENT FLAG directive to inform Task B that it may proceed. Task A then issues a DECLARE SIGNIFICANT EVENT directive to ensure that the Executive will schedule Task B.

Alternatively, Task A could issue a DECLARE SIGNIFICANT EVENT directive specifying event flag 35, which both sets the flag and declares a significant event.

Example 2

To synchronize the transmission of data between Tasks A and B, Task A specifies Task B and common event flag 42 in a SEND DATA directive.

Task B has specified flag 42 in a WAITFOR directive. When Task A's SEND DATA directive has caused the Executive to set flag 42 and to declare a significant event, Task B issues a RECEIVE DATA directive because its WAITFOR condition has been satisfied.

Example 3

A task contains a QUEUE I/O REQUEST and an associated WAITFOR directive, which both specify the same local event flag. When the task queues its I/O request, the Executive clears the local flag. If the requested I/O is incomplete when the task issues a WAITFOR directive that specifies the same local event flag, the Executive blocks the task.

When the requested I/O has completed, the Executive sets the local flag and causes an event. The task then resumes its execution at the instruction that follows the WAITFOR directive. The local event flag used in this manner ensures, for example, that the task does not attempt to manipulate incoming data until the transfer is complete.

Example 4

A task specifies the same local event flag in a MARK TIME and an associated WAITFOR directive. When the MARK TIME directive is issued, the Executive first clears the flag and subsequently sets it when the indicated time interval has elapsed.

If the task issues the WAITFOR directive before the flag has been set (that is, before the time interval has elapsed) the Executive blocks the task. The task then resumes when the Executive sets the flag.

Specifying an event flag does not imply that a WAITFOR directive must be issued. Event flag testing can be performed at any time. The purpose of a WAITFOR directive is to block task execution until an indicated event occurs. Hence, it is not necessary to issue a WAITFOR directive immediately following a QUEUE I/O REQUEST or a MARK TIME directive.

If a task issues a WAITFOR directive specifying an event flag that is already set, the blocking condition is satisfied and the Executive immediately returns control to the task.

The simplest way to test a single event flag is to issue the READ EVENT FLAG (RDEF$) directive, which can cause the following return codes:

IS.CLR—Flag was previously clear

IS.SET—Flag was previously set

The directives CLEAR EVENT FLAG (CLEF$) and SET EVENT FLAG (SETF$) also return the current state of the event flag before clearing or setting the flag.

For example, if a set common event flag indicates the completion of an operation, a task can issue the CLEF$ directive both to read the event flag and simultaneously to reset it for the next operation. If the event flag was previously clear (the current operation was incomplete), the flag remains clear.

The STOPFOR directives may also be used to wait for specified event flags. Use of these directives, rather than the corresponding WAITFOR directives, indicates that the task is prepared to be removed from memory, irrespective of task priority, until it is able to proceed. Section 4.2.2 further describes the effect of stopping a task.

2.4 System Traps

System traps are transfers of control (also called software interrupts) that provide tasks with a means of monitoring and reacting to events. The Executive initiates system traps when certain events occur. The trap transfers control to the task associated with the event and gives the task the opportunity to service the event by entering a user-written routine. The routine runs with the task's normal priority and privilege. The current PC and PS of the task is saved by the Executive when the service routine is entered. This enables the normal operation of the task to be resumed when the routine has completed its operation.

System traps fall into two distinct categories:

1 Synchronous System Traps (SSTs), which detect events directly associated with the execution of program instructions. They are "synchronous" because they always occur at the same point in the program when previous instructions are repeated. For example, an illegal instruction causes an SST. An SST can only occur while the task is already running. (That is, a task will never be made runnable just to service an SST).

2 Asynchronous System Traps (ASTs), which detect events that occur "asynchronously" to the task's execution; that is, the task has no direct control over the precise time that the event occurs. The completion of an I/O transfer might cause an AST to occur, for example. Unlike an SST, an AST can occur while the task is blocked (for example, waiting for an event flag). This might result in the task being given control of the processor simply to service the trap.

A task that uses the system trap facility issues system directives that establish entry points for user-written service routines. Entry points for SSTs are specified in a single table. AST entry points are set by individual directives for each kind of AST.

2.4.1 Synchronous System Traps (SSTs)

SSTs provide a means of servicing fault conditions within a task, such as memory protection violation and illegal instructions. These conditions, which are internal to a task and are not significant events, occur synchronously with respect to task execution. In these cases, if an SST service routine is not included in the task, the task's execution is aborted.

The user can set up an SST Vector Table, containing one entry per SST type. Each entry is the address of the SST routine that services the corresponding type of SST (the routine that services illegal instructions, for example). When an SST occurs, the Executive transfers control to the routine for that type of SST. If a corresponding routine is not specified in the table, the task is aborted. The SST routine enables the user to process the failure and then to return to the interrupted code.

An SST routine must be reentrant if that SST can occur within the SST routine itself. Although the Executive initiates SSTs, the execution of the related service routines is indistinguishable from the task's normal execution. An AST or another SST can therefore interrupt an SST routine.

2.4.2 SST Service Routines

The Executive initiates SST service routines by pushing the task's Processor Status (PS) and Program Counter (PC) onto the task's stack. The SST routine returns control to the task by issuing a Return from Interrupt (RTI) or Return from Trap (RTT) instruction. Note that the task's general registers R0-R5 are not saved; if the SST routine makes use of them, it must itself save and restore them.

SST routine execution is indistinguishable to the Executive from normal task execution. For example, all directive services are available to an SST routine. An SST routine can remove the interrupted PS and PC from the stack and transfer control anywhere in the task; the routine does not have to return control to the point of interruption. It should be noted that any operations performed by the routine (such as the modification of the Directive Status Word, or the setting or clearing of event flags) remain in effect when the routine eventually returns control to the task.

A trap vector table within the task contains all the service routine entry points. The user specifies the SST vector table by means of the SPECIFY SST VECTOR TABLE FOR TASK (SVTK$) directive. The trap vector table has the following format:

WD. 00 – Odd address or non-existent memory error
WD. 01 – Memory protect violation
WD. 02 – T-bit trap or instruction of a BPT instruction
WD. 03 – Execution of an IOT instruction
WD. 04 – Execution of a reserved instruction
WD. 05 – Execution of a non-IAS EMT instruction
WD. 06 – Execution of a TRAP instruction
WD. 07 – Synchronous floating point exception (PDP-11/40 only)
WD. 08 – Memory parity error

A zero appearing in the table means that no entry point is specified. If the corresponding SST occurs, the Executive aborts the task. The entry point or service routine specified for a particular condition may itself cause that condition (for example, the entry point specified for an odd address trap might be odd). In this case the Executive will repeatedly push the task PS and PC until the task stack overflows, causing the task to be aborted.

Depending on the reason for the SST, the task's stack might also contain the following additional information:

Memory protect violation

```
SP+10 -- PS
SP+06 -- PC
SP+04 -- Memory management status register (SR0)
SP+02 -- Virtual PC of the faulting instruction (SR2)
SP+00 -- Instruction backup register (SR1)
```

See the memory management unit section of the appropriate PDP-11 Processor Handbook for details of SR0, SR1 and SR2.

Trap instruction or EMT other than 377

```
SP+04 -- PS
SP+02 -- PC
SP+00 -- Instruction operand (low order byte) multiplied by 2,
         non-sign-extended
```

The additional information must be removed from the stack before the SST service routine exits (usually by means of an RTI or RTT instruction).

To include a debug vector table in the task, use the SPECIFY SST VECTOR TABLE FOR DEBUGGING AID (SVDB$) directive. The debug vector table has the same format as the trap vector table (described earlier in this section) but it contains addresses of entry points for SST routines for use by an intratask debugging aid (for example, ODT).

If both a task and a debugging aid table have been set up and a condition that causes an SST occurs, the Executive first checks the debugging aid table. If there is no entry point in that table, only then will the Executive check the task table. Thus, each entry in the debug vector table overrides the equivalent trap vector table entry if it is non-zero.

Either the RTI or the RTT instruction may be used to return from an SST service routine. In most situations their effect is equivalent but it is recommended that RTI be used in preference to RTT. The RTT instruction has a special use when a program is being traced using the trace facility and is fully described in the PDP-11 Processor Handbooks.

A task can change its condition codes by altering the saved PS on its stack before performing an RTI instruction. However, it cannot alter any of the other information in the PS (processor priority, processor mode, register set selection). These bits remain unaltered by the hardware when a user task performs an RTI or RTT instruction.

Figure 2–1 illustrates the task image and general flow of an SST.

2.4.3 Asynchronous System Traps (ASTs)

The primary purpose of an AST is to inform the task that a certain event has occurred. For example, a task can associate an AST with the completion of an I/O operation. When the AST informs the task that the event has occurred, the task can service the event and then return to the interrupted code.

Some directives can specify both an event flag and an AST; with these directives, ASTs can be used as an alternative to event flags or the two can be used together. This capability enables the user to specify the same AST routine for several directives, each with a different event flag. Thus, when the Executive passes control to the AST routine, the event flag can be used to determine the action required.

In contrast to the execution of an SST routine, which is indistinguishable from task execution, the Executive is aware that a task is executing an AST routine. An AST routine can be interrupted by an SST routine, but not by another AST routine.

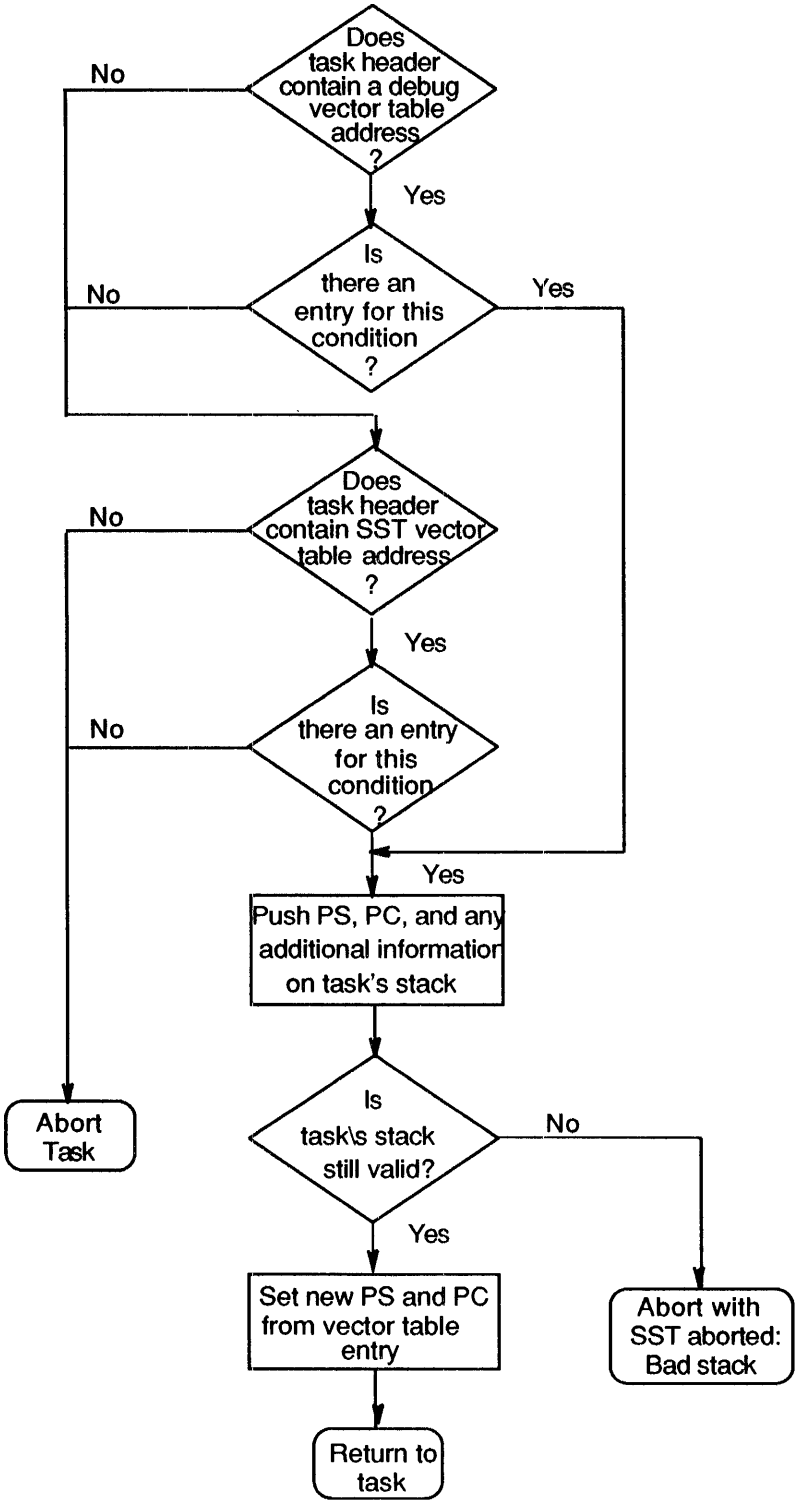
The following notes describe general characteristics and uses of ASTs:

1 If an AST occurs while the related task is executing, the task is interrupted to execute the AST service routine. When the AST service routine terminates, normal task execution continues unchanged unless the service routine has performed an operation which affects the task execution.

2 If an AST occurs while another AST is being processed, the Executive queues the latest AST (First-In-First-Out or FIFO) and then processes the next AST in the queue when the current AST service is complete (unless AST recognition was inhibited by the AST service routine). Only one AST for Receive Data and only one AST for Receive-by-Reference will be queued at any one time. In servicing the Receive Data or Receive-by-Reference, the program should attempt to receive until no more data or references are in the queue.

3 If a task is suspended when an associated AST occurs, the task remains suspended after the AST routine has executed, except in the following cases: the suspended task can be explicitly resumed by the AST service routine itself, by another task or by the PDS CONTINUE/REALTIME or MCR RESUME commands.

Figure 2–1 Task Image and Flow of an SST



4 If an AST occurs while the related task is waiting for an event flag setting (a WAITFOR directive), the task continues to wait after execution of the AST service routine unless the appropriate event flag has been set while the AST was being serviced (possibly by the AST service routine).

5 If an AST occurs for a checkpointed or swapped task, the Executive queues the AST (FIFO), and then effects it when the task is reloaded into memory. Exceptionally, RECEIVE (VRCD$/RCVD$) RECEIVE BY REFERENCE (RREF$) and powerfail recovery ASTs are handled in a fixed order, after any other ASTs which may have occurred while the task was checkpointed.

6 The Executive allocates the necessary nodes when an AST entry point is specified. Thus, an AST will never fail to be queued because of a lack of system nodes.

7 Two directives, INHIBIT AST RECOGNITION (IHAR$) and ENABLE AST RECOGNITION (ENAR$), allow ASTs to be queued for subsequent execution during the processing of critical sections of code. (A critical section might be one that accesses data bases also accessed by AST service routines, for example.) If ASTs occur while AST recognition is disabled, they are queued (FIFO) and then processed when AST recognition is enabled.

8 If an AST occurs while an SST is being processed, the SST service routine execution is not distinguished from task execution, and is interrupted for execution of the AST service routine.

9 An AST routine may not be executed immediately when the AST occurs if, for example, a higher priority process is running. In this case, one or more ASTs may occur before the first AST routine is executed. This can cause synchronization problems if the AST routines are interdependent.

2.4.4 AST Service Routines

When an AST occurs, the Executive pushes the task's event flag mask words, the DSW, the PS, and the PC onto the task's stack. This in effect saves the state of the task so that the AST service routine may make use of Executive services without affecting the operation of the interrupted code. Depending on the reason for the AST, the stack may also contain additional parameters. Note that the task's general registers R0-R5 are not saved; if the routine makes use of them, it must itself save and restore them. Failure to do this will cause unpredictable errors in the main task code and is one of the most common causes of errors in programs which use ASTs.

The event flag mask words indicate the event flag(s) for which the task is waiting or stopped, corresponding to the WAIT FOR LOGICAL OR OF FLAGS (WTLO$), WAIT FOR SIGNIFICANT EVENT (WSIG$), STOP FOR LOGICAL OR OF EVENT FLAGS (STLO$), or STOP FOR SINGLE EVENT FLAG (STSE$) system directives. Their actual meaning and value is not defined. In particular, the effect of changing them is unpredictable.

After processing an AST, the task must remove the trap-dependent parameters from its stack. It must then issue an AST SERVICE EXIT (ASTX$) directive with the stack set as indicated in the description of that directive (see the IAS System Directives Reference Manual). When the AST service routine exits, it returns control to the original task. However, if any further ASTs are queued, the first is serviced immediately (unless AST recognition has been inhibited) and the original task never actually gains control.

Task stack format can occur with the following six variations:

1 If a task needs to be notified when a Floating Point Processor exception trap occurs, it issues a SPECIFY FLOATING POINT EXCEPTION AST (SFPA$) directive. If the task specifies this

directive, an AST will occur when a Floating Point Processor exception trap occurs. The stack will contain the following values:

SP+20—Event flag mask word for flags 1 to 16
SP+16—Event flag mask word for flags 17 to 32
SP+14—Event flags mask word for flags 33 to 48
SP+12—Event flag mask word for flags 49 to 64
SP+10—PS of task prior to AST
SP+06—PC of task prior to AST
SP+04—Task's Directive Status Word
SP+02—Floating exception code
SP+00—Floating exception address

2 If the task needs to be notified of power failure recoveries, it issues a SPECIFY POWER RECOVERY AST (SPRA$) directive. If the task specifies this directive, an AST will occur when the power is restored. The stack will contain the following values:

SP+14—Event flag mask word for flags 1 to 16
SP+12—Event flag mask word for flags 17 to 32
SP+10—Event flag mask word for flags 33 to 48
SP+06—Event flag mask word for flags 49 to 64
SP+04—PS of task prior to AST
SP+02—PC of task prior to AST
SP+00—Task's Directive Status Word

3 If a task needs to be notified when it receives either a message or a reference to a shareable area, it issues either a SPECIFY RECEIVE AST (SRDA$) or a SPECIFY RECEIVE-BY-REFERENCE AST (SRRA$) directive. If the task specifies one of these directives, an AST will occur when a message or reference is sent to the task. An AST also occurs when a task has at least one item in the receive queue when the task is checkpointed into or initially loaded into memory. The stack will contain the following values:

SP+14—Event flag mask word for flags 1 to 16
SP+12—Event flag mask word for flags 17 to 32
SP+10—Event flag mask word for flags 33 to 48
SP+06—Event flag mask word for flags 49 to 64
SP+04—PS of task prior to AST
SP+02—PC of task prior to AST
SP+00—Task's Directive Status Word

4 When a task queues an I/O request and specifies an AST service entry point, an AST will occur upon completion of the I/O request. The task's stack will contain the following values:

SP+16—Event flag mask word for flags 1 to 16
SP+14—Event flag mask word for flags 17 to 32
SP+12—Event flag mask work for flags 33 to 48
SP+10—Event flag mask word for flags 49 to 64
SP+06—PS of task prior to AST
SP+04—PC of task prior to AST
SP+02—Task's Directive Status Word
SP+00—Address of I/O status block for I/O request (or zero if none was specified).

5 When a task issues a MARK TIME (MRKT$) directive and specifies an AST service entry point, an AST will occur when the indicated time interval has elapsed. The task's stack will contain the following values:

SP+16—Event flag mask word for flags 1 to 16

SP+14—Event flag mask word for flags 17 to 32
SP+12—Event flag mask word for flags 33 to 48
SP+10—Event flag mask word for flags 49 to 64
SP+06—PS of task prior to AST
SP+04—PC of task prior to AST
SP+02—Task's Directive Status Word
SP+00—Event flag number (or zero if none was specified)

6 When a task issues a SPAWN (SPWN$) directive and specifies an AST service entry point, an AST will occur when the specified task terminates. The task's stack will contain the following values:

SP+16—Event flag mask word for flags 1 to 16
SP+14—Event flag mask word for flags 17 to 32
SP+12—Event flag mask word for flags 33 to 48
SP+10—Event flag mask word for flags 49 to 64
SP+06—PS of task prior to AST
SP+04—PC of task prior to AST
SP+02—Task's Directive Status Word
SP+00—Exit status block address

ASTs can also be caused by other privileged tasks. For example, the IAS terminal handler and IAS DECNET may both cause ASTs under certain circumstances. Check the appropriate documentation to determine the state of the stack on entry to AST service routines.

Figure 2–2 shows a typical sequence of events when an AST occurs.

2.5 Intertask Communications

Tasks frequently need to transfer data between one another. Figure 2–3 shows an example of a task using an event flag for intertask communication.

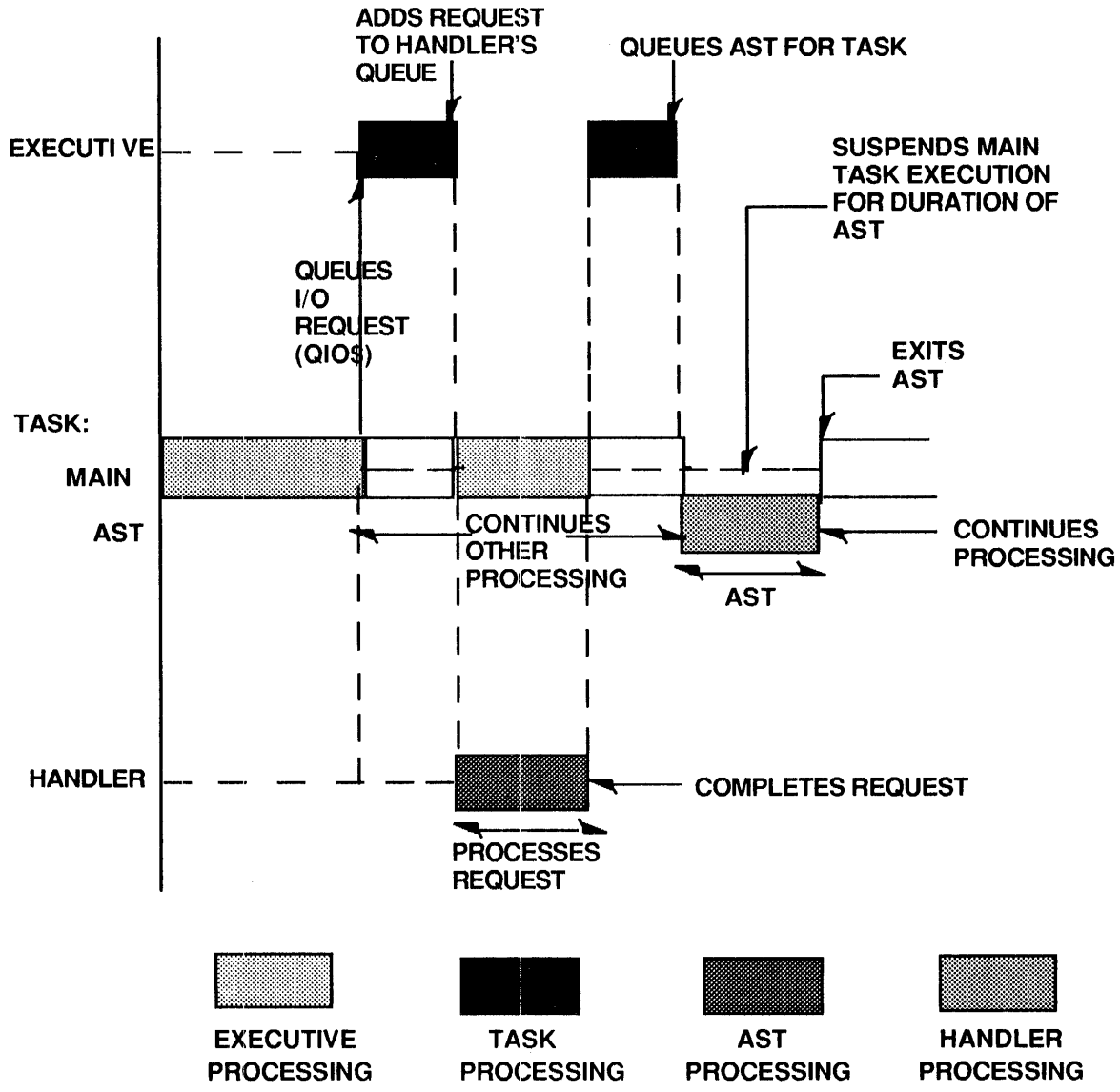
Five techniques for intertask communication and data transfer are provided.

- Event flags
- Shareable Global Areas (SGAs)
- Dynamic regions
- SEND and RECEIVE data blocks
- Shared access to data files.

2.5.1 Event Flags

Event flags are fully described in Section 2.3. These flags are generally used for synchronization purposes in conjunction with one of the other techniques described in Section 2.5.

Figure 2–2 Sample Sequence of Events for an AST



2.5.2 Shareable Global Areas

Shareable Global Areas (SGAs) are independent units of code and/or data that can be accessed by more than one task concurrently. SGAs can be used for intertask communication because they allow several tasks to access, and hence communicate through, a common read/write data area. For example, a task might read information from an I/O device, store it in an SGA, and set a global event flag to notify another task that data is available for use in that SGA.

This technique is useful when tasks need to communicate large amounts of data. No physical movement of the data takes place and only a single data area is needed.

Figure 2–3 Using an Event Flag for Intertask Communication

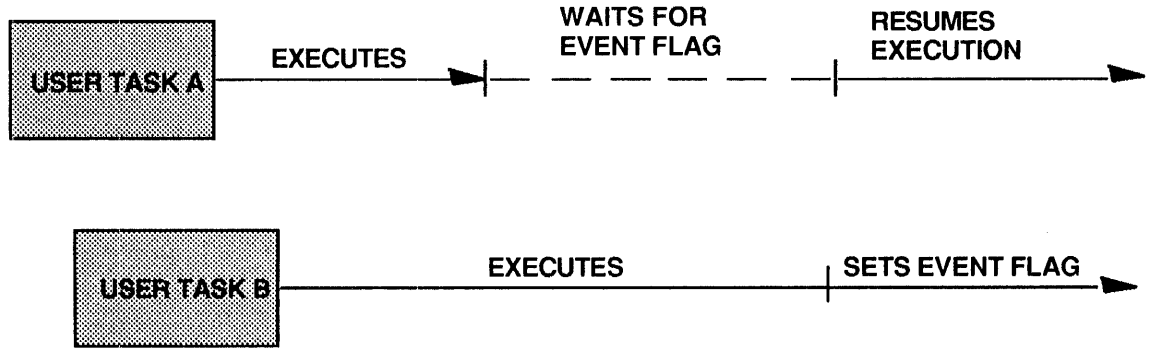
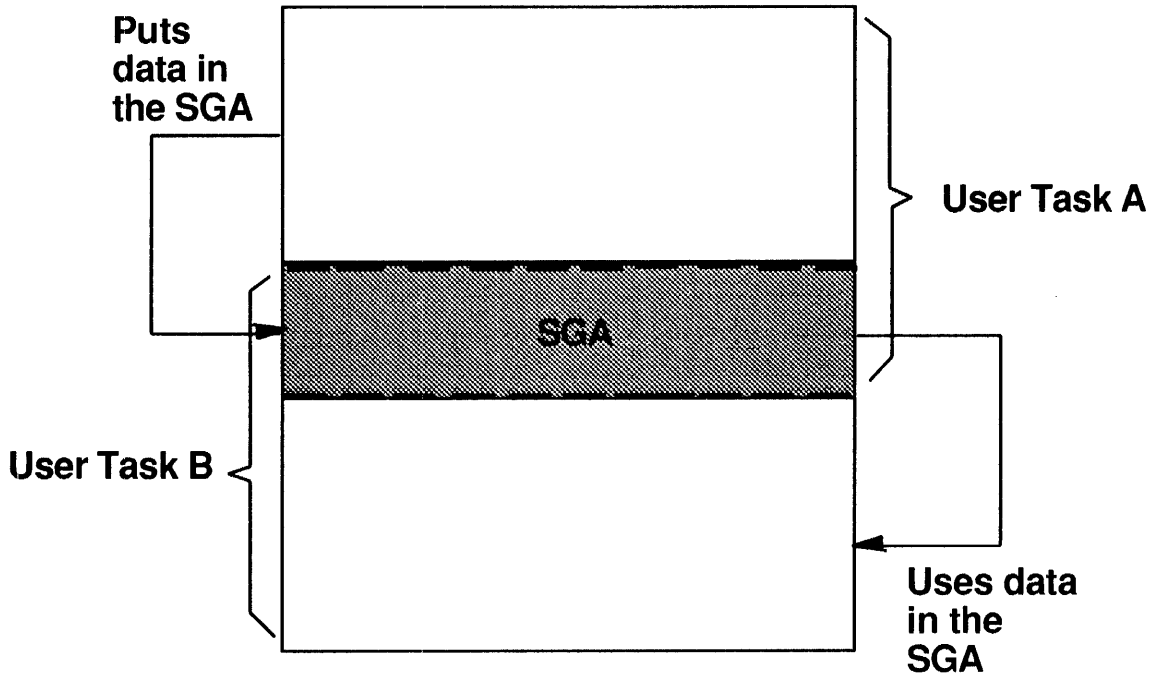


Figure 2–4 Using an SGA for Intertask Communication



Shareable Global Areas are described more fully in Chapter 5. Figure 2–4 illustrates the use of an SGA for intertask communication.

2.5.3 Dynamic Regions

Dynamic regions are similar to SGAs in that they allow two or more tasks to access a single data area. However, dynamic regions provide greater flexibility because they can be created dynamically by a task. Only one copy of an SGA can exist in the system at one time, so that only one invocation of a group of interacting tasks which communicate via the SGA can run. In contrast, one task of the group can create a dynamic region and send access to that region to each of the other tasks by using the SEND BY REFERENCE (SREF$) or SEND BY REFERENCE AND RESUME OR REQUEST (SRFR$) directive.

Dynamic regions can also be used to provide a variant of the SEND DATA facility. One task can create a region and fill that region with data, then use the send-by-reference facility to send the data to another task. The task can then detach the region and start afresh. This is the most efficient means of sending large amounts of data, generated by one task, to another task. No SCOM nodes are used, the remapping to "receive" the data is quick, and no physical transfer of data occurs.

Chapter 2 of the *IAS System Directives Reference Manual* fully describes how you can manipulate dynamic regions by using the memory management system directives.

2.5.4 SEND and RECEIVE Data Blocks

Tasks can communicate by transfering blocks of data between one another. The receiving task need not be active or in memory at the time of the transfer. The data blocks are queued for the task, and can then be received by the task when the task is activated. The data blocks transferred can be up to 255 (decimal) words in length. The following SEND/RECEIVE directives are provided in IAS to enable transfer of data blocks from one task to another:

- SEND DATA (VSDA$/SDAT$), which queues a data block, by priority, for a task to receive.
- SEND DATA AND RESUME OR REQUEST RECEIVER (VSDR$/SDRQ$), which performs as for SEND DATA except that the receiving task can be additionally resumed or requested.
- RECEIVE DATA (VRCD$/RCVD$), which enables a task to receive a data block queued by another task.
- RECEIVE DATA OR EXIT (VRCX$/RCVX$), which performs as for RECEIVE DATA except that the task will exit if no data is queued.
- RECEIVE DATA OR SUSPEND (VRCS$/RCVS$), which also performs as for RECEIVE DATA except that the task will be suspended if no data is queued.
- RECEIVE DATA OR STOP (VRCT$/RCST$), which again performs as for RECEIVE DATA except that the task will be stopped if no data is queued.

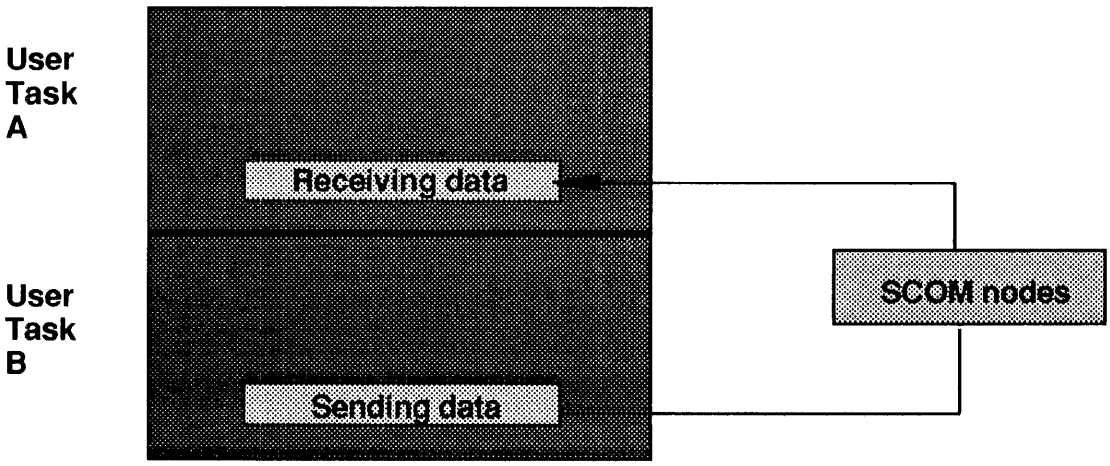
The following information should also be noted with reference to the SEND and RECEIVE data facility:

- Variable length data up to 255 words can be sent and received.
- If the receiver's buffer is too small to hold all the data sent, the excess is lost and the receiver is notified by an error code in the DSW.
- A task is identified by its name (up to 6 alphanumeric characters) and, if it is a multiuser task, by the device for which the task was initially requested (its TI). If the task was initiated by another task, the TI of the requesting task becomes the TI of the requested task also, unless a different TI was specified.

- Multiuser tasks issuing receives are passed only that data sent with the same TI as the multiuser task. This approach ensures the proper flow of data among several multiuser tasks and a single-user task when the single-user task is receiving from and sending to the multiuser tasks.
- A task can determine its own TI by sending data to itself without specifying a TI, and then receiving that data specifying a location to store the sender's TI.

See the *IAS System Directives Reference Manual* for a detailed description of each system directive. Figure 2–5 shows an example of a SEND/RECEIVE directive being used for intertask communication.

Figure 2–5 Using a SEND/RECEIVE Directive for Intertask Communication



The SEND DATA facility is ideally suited to transferring small amounts of data between tasks. Where large amounts of data are involved, one of the other methods described in Section 2.4 is normally more efficient.

2.5.5 Shared Data Files

Shared data files provide the same data communication facility as shareable global areas except that they are maintained on a FILES-11 volume rather than in main memory. This permits a much greater quantity of data to be shared and provides greater security because the data is always stored on disk, where it will be safe if the system should fail for any reason. On the other hand, data files are much less efficient because every reference requires a disk access.

Special considerations apply to file access when several tasks are simultaneously accessing a file for reading or writing. The *IAS I/O Operations Reference Manual* and the *IAS RMS-11 MACRO Programmer's Reference Manual* contain full descriptions of file access using the facilities of the File Control Services (FCS) and RMS-11 respectively.

3 System Data Structures

This chapter describes the organization of the system data structures. The following are described in this chapter:

1 The formats of fixed-length tables and linked lists (see Section 3.1).

2 The use of nodes for node accounting (see Section 3.2).

3 The general layout of the system common areas (see Section 3.3).

4 The standard offsets for nodes within the System Communication Area (SCOM) (see Section 3.3.5).

5 The most significant aspects of each data structure within SCOM and the IAS Common Area (IASCOM). See Sections 3.4 through 3.25.

6 The most significant aspects of the Task Header data structure (see Section 3.26).

The intention of this chapter is to give a real-time programmer an understanding of obscure problems in programs and sets of interacting programs, based on information readily available while the system is running.

The complete contents of each data structure are listed in Appendix A, with some explanation of their content.

It is not the intention of this chapter to describe every field within each data structure. Rather, where specific fields have particular importance to the execution of real time tasks, the reasons are highlighted and, in many cases, the fields are described in depth.

3.1 Fixed-Length Tables and Linked Lists

Many of the system data structures are contained in fixed-length tables and linked lists. A fixed-length table comprises information which has been specified at system generation time (for example, information about partitions). Linked lists are double-ended queues, or *deques* (pronounced "decks"), designed to enable list elements to be added or deleted from anywhere in the list by means of backward and forward pointers.

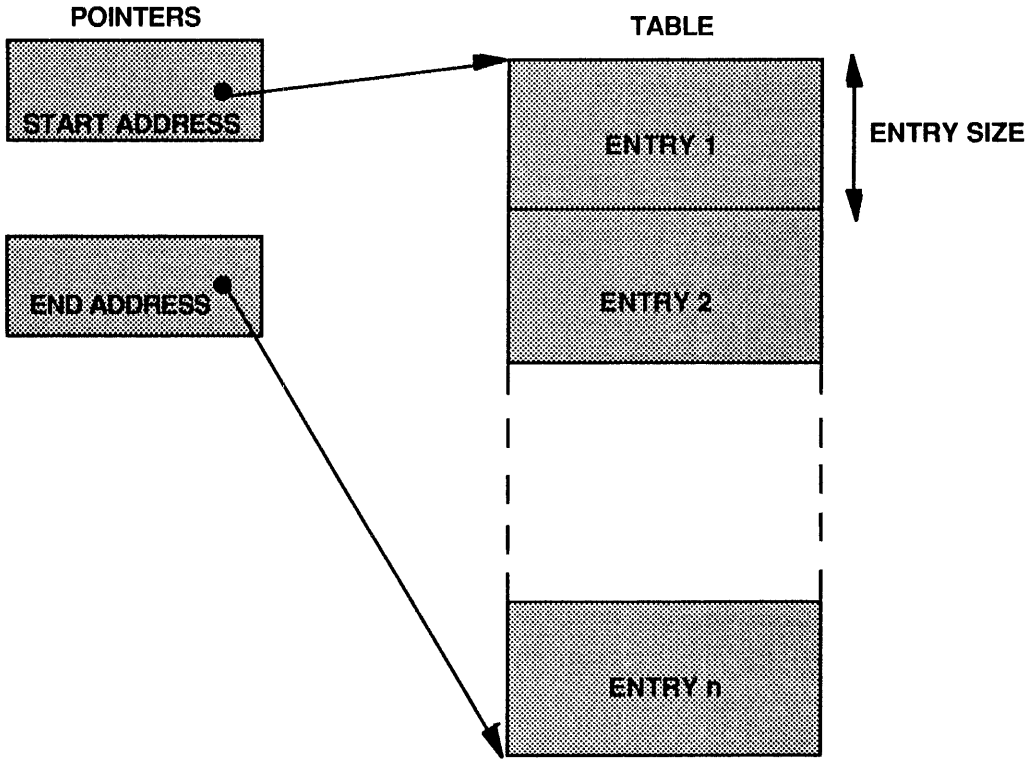
3.1.1 Accessing Fixed-Length Tables

A fixed-length table resides in consecutive memory locations. The entries which constitute each fixed table are not necessarily ordered in any special sequence in memory. However, the size of each fixed table is determined at system generation and cannot be changed without performing another system generation. This format is used when the list is static, when scan time is critical, or both.

Fixed-length tables are accessed by means of start and end address pointers. The end address points to the word immediately after the last word of the last entry in the table.

Figure 3–1 illustrates the format of a fixed-length table.

Figure 3–1 Format of a Fixed-Length Table



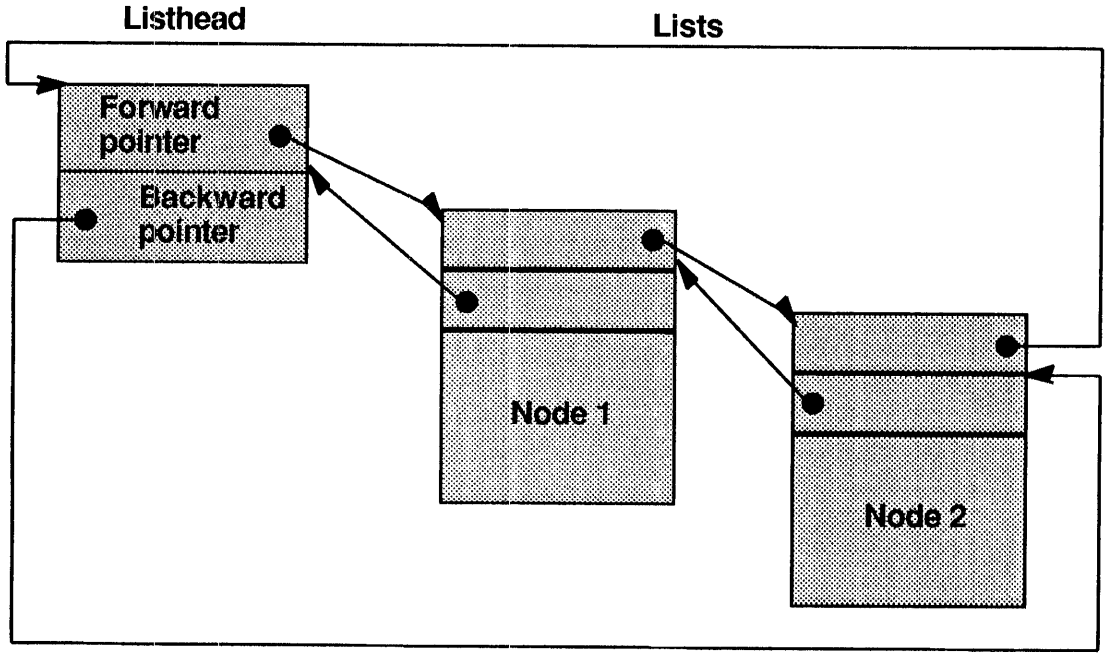
3.1.2 Accessing Linked Lists

A linked list, or deque, is a circularly linked list consisting of list elements called nodes. The first word of each node points forward to the next node. The second word of a node is a backward pointer to the first word of the previous node. The addresses of the first and last nodes in the list are contained in a two-word listhead. This is linked into the list, by forward and backward pointers, exactly as though it were just another node.

Forward and backward pointers enable nodes to be inserted or deleted from any part of the list. The nodes are accessed by setting a pointer to the listhead and obtaining the address of the next node from the first word of the current node. This operation is performed until the forward pointer is found to be pointing back to the listhead. If the first word of the listhead points to itself, the deque is empty.

Figure 3–2 illustrates the format of a linked list.

Figure 3–2 Format of a Linked List



3.2 Node Accounting

Node accounting is performed in eight-word blocks. When a task issues a system directive which requires nodes, it will be charged for the number of eight-word blocks used. For example, the QIO$ directive requires a 24-word I/O request node, so if the request is queued successfully the task will be charged for three eight-word nodes.

Three parameters determine whether a task can obtain nodes:

1 The node pool utilization limit.

2 The node pool usage count.

3 The active versions count (used only for multiuser tasks).

3.2.1 Node Pool Utilization Limit

This is the maximum number of eight-word nodes that can be picked by each active version of the task. The limit is established when a task is installed. It is set to a value between 0 and 255 nodes when the task is built or installed. If a value is not specified, the limit defaults to 40 nodes. Except for special purposes, it is recommended that a value no less than 40 should be specified.

3.2.2 Node Pool Usage Count

This is the total number of 8-word nodes currently charged to the task, for all active versions. When a task requires nodes, normally as a result of issuing a system directive, the node pool usage count is compared with the node pool utilization limit. If the limit will be exceeded, the directive is rejected with an 'unavailable pool node' error. If, however, the limit will not be exceeded, the nodes are charged to the task and the following occurs:

1 The node pool usage count is incremented by one unit for each eight-word node.

2 The nodes are made available to the task so that the directive can be performed.

3.2.3 Active Versions Count

The node pool utilization limit is a limit per active version of a task. When more than one version of a task is active, the effective limit is the product of the number of active versions and the limit per version. The node pool usage count is a count of the nodes used by all versions of the task. For example, if the node pool utilization limit of a task is 40 and there are two active versions of that task, the two tasks can use a total of 80 nodes.

The system does not check for an even distribution of nodes between active versions of the same task. Therefore, in this example, it is possible that one active version could use 70 nodes, limiting the other active version to 10 nodes. If however, the version of the task with 10 nodes exits, the other version will be over its limit. The over-limit version will not be able to pick any further nodes until at least 30 nodes have been returned to the pool.

These three quantities described in Sections 3.2.1 to Section 3.2.3 are all contained in the task's STD node (see Section 3.5). The actual algorithm used to determine whether a task might have the nodes it requires is:

```
limit x AV > usage count + requirement
```

where:

- AV = Active versions count, except that if this is zero a value of 1 is used. This enables nodes to be picked on behalf of an inactive task (for example, because of a RUN$ request that becomes due after the task has exited).

See the *IAS System Directives Reference Manual* for further information about the way nodes are used for directive processing.

3.3 System Data Structures

The system must hold information which describes all aspects of its current operations. This information is referred to as the System Data Structures. All Executive operations result in some reference to or manipulation of the System Data Structures. The system stores data structures in one of three storage areas:

1 The System Common Areas (SCOM and IASCOM).

2 Task headers.

3 Free memory.

Of the three storage areas, the System Common Areas are most often used. Task headers and free memory are used in specific circumstances to reduce the use of the System Common Areas. The System Common Areas compose two major data areas:

1 The System Communication Area (SCOM).

2 The IAS Common Area (IASCOM).

Task headers are described in Sections 3.3.6 and 3.26.

3.3.1 System Communication Area (SCOM)

The Executive and privileged tasks map onto the System Communication Area (SCOM). This area consists of a number of fixed tables or lists and subroutines, with the remaining space being available in eight-word blocks known as nodes. These nodes are used by the system for various purposes, including intertask communication.

SCOM contains two regions:

1 System data and system subroutines.

2 System data structures.

Figure 3–3 shows an example of Executive and task mapping onto SCOM.

3.3.2 Summary of SCOM Data Structures

SCOM contains the following data structures:

1 The Active Task List (ATL)—A priority-ordered deque of all active tasks (see Section 3.4).

2 The System Task Directory (STD)—A directory of all tasks installed in the system (see Section 3.5).

3 The Physical Unit Directory (PUD)—A table of entries for each physical device unit defined in the system (see Section 3.6).

4 The Task Partition Directory (TPD)—A table of entries for each partition defined in the system (see Section 3.7).

5 The Global Common Directory (GCD)—A linked list of entries for shareable global areas and regions (see Section 3.8).

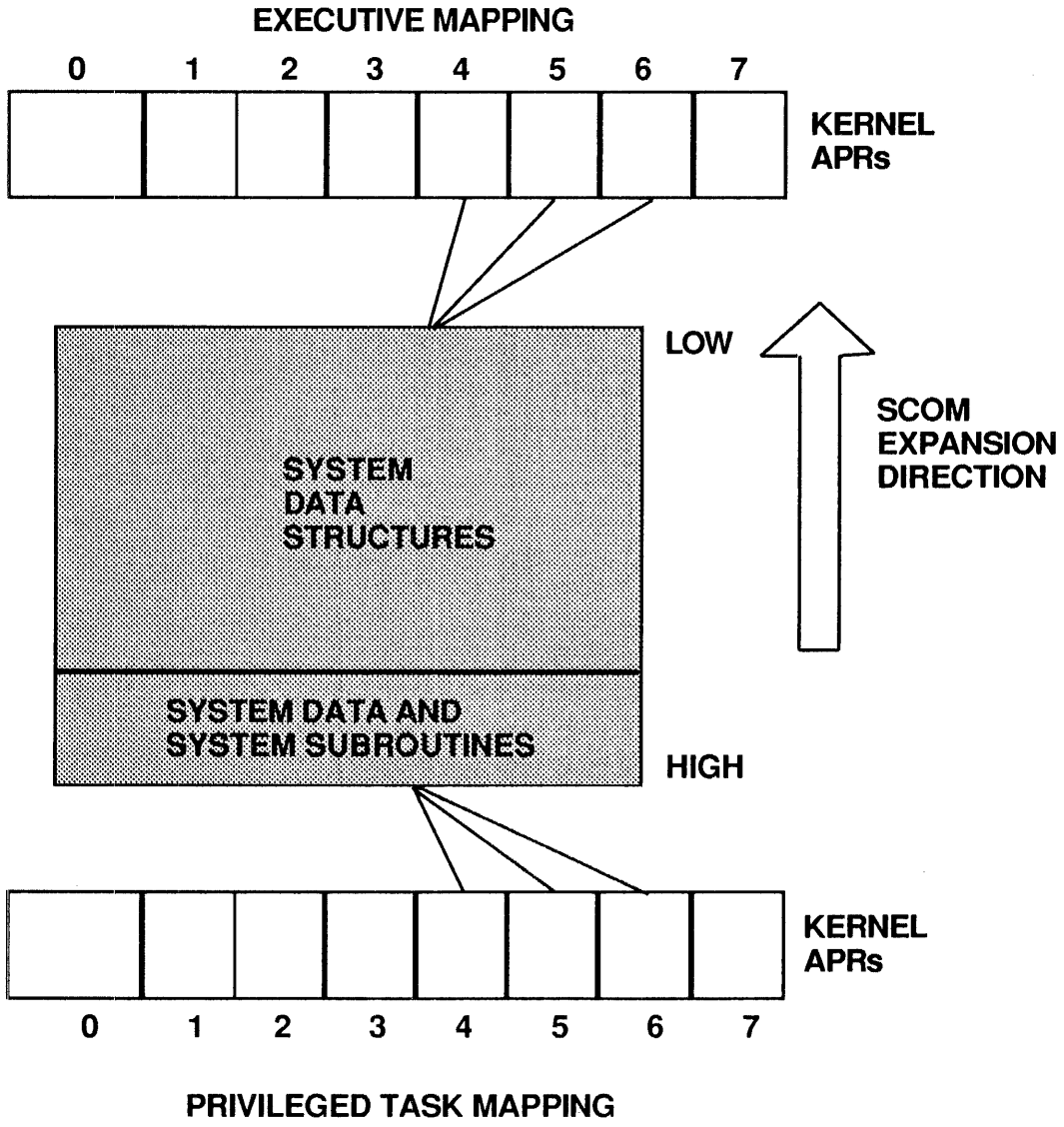
6 The Input/Output Request Queues (IRQ)—The queues of input/output requests for physical devices (see Section 3.9).

7 The Clock Queue (CKQ)—A list with one entry for each operation to be performed at some future time (for example, starting the execution of a scheduled task). See Section 3.10.

8 The Asynchronous System Trap Queues (ASQ)—Lists containing one node for each Asynchronous System Trap (AST) to be executed. See Section 3.11).

9 The SEND/RECEIVE Queues (SRQ)—Lists consisting of one node for each block of data to be sent to a task (see Section 3.12).

Figure 3–3 Executive and Task Mapping to SCOM



10 The SEND/RECEIVE by Reference Queue (RRQ)—A single list containing all data packets for all SEND/RECEIVE-by-reference information (see Section 3.13).

11 The Spawned Task List (STL)—A list containing one node for each spawned task (see Section 3.14).

12 The User Task List (UTL)—A list containing information used by the IAS scheduler (see Section 3.15).

13 The Swap File List (SFL)—A list containing one node for each swap file currently available in the system (see (see Section 3.16).

14 The Memory Usage Lists (MUL)—The lists that control the allocation of memory within a timesharing partition (see (see Section 3.17).

15 The Fixed Task List (FTL)—A deque of nodes for each task fixed in memory (see (see Section 3.18).

3.3.3 IAS Common Area (IASCOM)

The IAS Common Area (IASCOM) contains the data structures that are specifically required for timesharing activities in the system. IASCOM consists of two regions:

1 A communications region containing all the listheads, global symbol data, and scheduling details.

2 A variable-sized area containing the tables and node pools required for the IAS data tables.

3.3.4 Summary of IASCOM Data Structures

IASCOM contains the following data structures:

- The User Job Nodes (UJN)—Each node contains all information about a scheduler controlled task not contained in the ATL node (see Section 3.19).
- The User Terminal Nodes (UTN)—Each node contains the timesharing device characteristics and information regarding the current activities for the terminal (see Section 3.20).
- The Command Interpreter Table (CIT)—A table containing one node for each CLI in the system (See Section 3.21).
- The Device Table (DVT)—A table that supplements the information contained in the PUD and contains device usage information for timesharing users (See Section 3.22).
- The Device Load Table (DLT)—A table that contains one node for each device mounted in the system (see Section 3.23).
- The Job Node Pool (JNP)—A pool for currently unused job nodes (see Section 3.24).
- The Terminal Node Pool (TNP)—A pool for currently unused terminal nodes (see Section 3.25).

3.3.5 Standard Offsets for SCOM Data Structures

To enable system subroutines to perform standard operations on linked lists, certain information is stored in nodes at standard offsets. The following standard offsets occur frequently in SCOM data structures:

- Forward and Backward Pointers (offsets N.FP/N.BP).

  Words 0 and 1, respectively, of all nodes contained in linked lists are forward and backward pointers. These pointers are used by system subroutines to manipulate linked list data structures (for example, inserting or deleting a node from a list).

- Node Accounting Word (offset N.AW).

  This word contains the STD address of the task which is charged for the use of a node. When a task returns a node to the node pool, the executive takes the requesting task's STD address from offset N.AW so that it can decrement the task's node pool usage count. This information is often useful for other purposes (for example, the requestor of a task in its ATL node), and is frequently referenced by another name.

- TI Indicator (offset N.TI).

  This indicator contains the PUD address of the terminal from which a task was initiated and is used to distinguish between different versions of a multiuser task. For example, offset N.TI is used when a list contains SEND/RECEIVE nodes for all versions of a task but the node required from the list is for a particular copy of the task.

- Priority (offset N.PR).

  This offset contains the current priority of the entity described by the node. This is used by the system subroutine ..IPRI when nodes are to be inserted into a list in priority order (so that the highest priority node will be serviced next).

- Status (offset N.SB).

  This offset defines the current state of a task or Shareable Global Area (SGA). This is used when the status of tasks or SGAs are processed by the system subroutine ..IODN, which performs I/O request completion.

3.3.6 Task Headers

The task header data structure is used mainly to hold information about the current state of a task. This information could be held in SCOM nodes if there were sufficient space. However, because of limitations of virtual address space and physical memory, the information is held in a task header, which is copied to disk when the task is not in memory. The information is stored immediately before the main task image in the task image file so that the following actions occur:

1 The header is automatically loaded with the task code.

2 System components can access the information easily.

The task header is used mainly by the following facilities:

- Task Builder
- INSTALL task
- Executive

The task builder initializes the contents of the header in the task image disk file. The task builder decides how big the header has to be, and sets up locations such as the initial PC (the program start address).

INSTALL reads the header from the disk, fills in various fields, and writes the header back to the disk file. In particular, INSTALL fills in static information regarding a task's SGAs.

The Executive makes the most frequent use of the task header. The Executive references and modifies the header during the following operations:

- Loading tasks and their regions
- Running the tasks

When loading the task, the Executive converts much of the information in the header to reflect the task's new state. Thus, for example, the Executive converts information about a task's regions from pointers to the regions' GCD nodes to real physical addresses for the regions. The Executive automatically saves the information in the header on disk when the task is swapped or checkpointed. The information is read back into memory when the task is reloaded.

3.4 Active Task List (ATL)

The system coordinates scheduling of all system tasks by scanning entries in a priority-ordered list of tasks called the Active Task List (ATL). Each entry in the list is a node containing execution characteristics of an active task. The ATL has an entry for every active task in the system. It also uses dummy entries to control parts of the Executive (for example, the system null tasks).

Section 4.2.1 describes how ATL is used to control task scheduling. Fields of the ATL that are used only for tasks under the control of the IAS scheduler are not described here.

Use the SHOW TASKS/ACTIVE (PDS) or ACT (MCR) command to display a summary of the ATL or to give full information about a particular entry. See the *IAS PDS User's Guide* and the *IAS MCR User's Guide* for details about these commands.

3.4.1 Terminal Identification (A.TI)

The PUD address of the terminal where the task is running is held at standard offset N.TI.

3.4.2 Run Priority (A.RP)

This is the current run priority of the task. Since the ATL is linked in priority order, this is held at standard offset N.PR.

3.4.3 I/O Pending Count (A.IN)

This is a count of I/O requests that have been queued by the task but have not yet been completed. The count is also incremented for each opened file and attached device, and by certain device handlers that must be informed when the task exits.

The count is used when the task exits. If it is non-zero, the Executive performs "I/O rundown" to terminate all pending I/O, to make the count zero.

3.4.4 Saved Status of Checkpointed Task (A.CS)

When a task is checkpointed, the Executive uses the task status to control the checkpointing/reloading process. The task's actual status must be saved so that it can be restored when the task is reloaded.

3.4.5 Task's I/O in Progress Count (A.IR)

Each task in the system has a count of the number of I/O operations currently in progress. When an I/O operation is dequeued, the count is incremented, and when the operation is completed the count is decremented. If a task is built checkpointable, it can be checkpointed only if no unswappable I/O is currently in progress (that is, the count must be equal to the swap I/O count). See Section 3.4.17 for details of the swap I/O count. See Section 4.2.4 for further details about checkpointing.

3.4.6 Task's Mark Time Pending Count (A.MT)

This is a count of all mark time requests issued by the task that have not yet come due. It is used when the task exits, to determine whether it is necessary to scan the Clock Queue to remove mark-time requests. See Section 3.10 for a description of the Clock Queue.

3.4.7 Saved Checkpoint Priority (A.CP)

When a task is to be checkpointed, so that the checkpoint operation is satisfied as rapidly as possible, the checkpointed task temporarily assumes the higher priority of the task requiring the memory. When the task has been checkpointed, the system restores the priority of the task from the checkpoint priority to the original priority.

3.4.8 Real Address of Load Image (A.HA)

This is the base address of the task load image (actually the base of the header). It is held as a 32-word block number, in the form required by the memory management hardware.

3.4.9 Task State (A.TS)

Every entry in the ATL contains a task state. The Executive uses the state to determine the action to take when processing a task's ATL entry. For example, if the state is "WF0" (waiting for event flags 1-16), the Executive checks to see whether any of the flags being waited for are set, and if so changes the state to "RUN" (runnable). Table 3–1 shows the meaning of each task state.

NOTE: Most states are transient; that is, a task will move on to another state within a fraction of a second. States marked "*" are longer lived and a task can quite reasonably stay in one of these states indefinitely.

Table 3–1 Task States

AST	Task has an AST queued and is about to enter an AST service routine.
DIF	"Directive Finished." Task was waiting for a directive to complete, and the directive has now completed.
EXT	"Exited." The Executive will perform the clean-up operations described under the EXIT directive in the *IAS System Directives Reference Manual*, then deallocate the ATL node.
IDL	"Idle." This is a special state used to trigger the null task. No normal task ever has this state.
IR1 IR2 IR3 IR4	These states all indicate that the task is having I/O rundown performed. I/O rundown is described in Chapter 6.
LRF	"Load request failure." A load failure has occurred while loading the task from disk.
LRG	"Load request for global area." Task is waiting for a global area or dynamic region to be loaded.
LRP	"Load request pending." The task is about to be loaded from disk.
LRQ	"Load request queued." Task is being loaded from disk.
LRS	"Load request successful." Task has been successfully loaded from disk. The Executive will perform any necessary initialization, then change the state to "RUN" to enable the task to commence or resume execution.

Table 3–1 (Cont.) Task States

MEX	"Marked for extension." The Executive is in the process of adjusting the task size in memory and allocating new swap space of the correct size.
MRE	"Memory required for execution." Task is being fixed or was requested using EXEC$, and the system is trying to find memory for the task.
*MRL	"Memory required for load." Task is waiting for memory to be allocated.
*MRR	"Memory required for region." Task is waiting for memory to be allocated for a shareable global area or dynamic region.
*PAR	"Parity error." Task has terminated because of a memory parity error. The task is left in this state, and its memory left allocated, so that the faulty memory is not reused.
RLA	"Reload for AST." Task is being reloaded (while stopped) to determine whether it has declared an AST for receive data, receive-by-reference, or powerfail.
RRF	"Record request failure." A transfer error has occurred while writing the task to disk (as a result of checkpointing or swapping).
RRQ	"Record request queued." Task is being written to disk.
RRS	"Record request successful." Task has been successfully written to disk.
*RUN	"Runnable." Task is runnable, and can be given control of the processor.
SFC	"Suspended for checkpointing." Task is about to be checkpointed and is waiting for non-swappable I/O requests to complete.
*ST0	Task is "stopped" for event flags in the range 1-16.
*ST1	Task is "stopped" for event flags in the range 17-32.
*ST2	Task is "stopped" for event flags in the range 33-48.
*ST3	Task is "stopped" for event flags in the range 49-64.
*ST4	Task is "stopped" for event flags in the range 1-64.
*STN	"Suspended for termination notice;" that is, waiting for the .TKTN. task to print a "TASK ABORTED..." message.
*SUS	Task is suspended.
*STP	Task is stopped.
TFF	"Termination for fault." Task has been aborted because of a fault (for example, odd address).
TNR	"Termination notice requested." Task is waiting for .TKTN. to be requested. When .TKTN has been successfully requested, the state is changed to STN.
TS1 TS2	Special states used to trigger the IAS scheduler. No normal task ever has these states.
TSE	"Timesharing exit." Task under control of IAS scheduler has exited.
WDI	"Waiting for directive." Task is waiting for a directive (EXEC$,FIX$) to be completed.
*WF0	Task is waiting for event flags in the range 1-16.
*WF1	Task is waiting for event flags in the range 17-32.
*WF2	Task is waiting for event flags in the range 33-48.
*WF3	Task is waiting for event flags in the range 49-64.
*WF4	Task is waiting for event flags in the range 1-64.
WND	"Waiting for nodes." Task is waiting for enough nodes to become available to perform a directive.
*WSM	"Waiting for semaphore." Task is a privileged task waiting for exclusive access to an Executive data structure.

3.4.10 AST Indicator (A.AS)

When a task is processing an AST, the task's pre-AST state is stored so that it can be restored when the task exits from its AST service routine.

3.4.11 STD Entry Address (A.TD)

All installed tasks have an STD entry address. When a task is active, the ATL node has an STD entry address which identifies which task is running.

3.4.12 Task's Event Flags (A.EF)

These two words contain the task's local event flags, 1-32. Event flag 1 is represented by bit 0 of the first word; event flag 17 is represented by bit 0 of the second word.

3.4.13 Task's Event Flag Masks (A.FM)

The event flag masks at A.FM are used for various purposes by the Executive, to record information supplementary to the state of a task. The significance of these words depends on the state of the task, as follows.

1 Up to first time load (states LRP, LRQ, LRS):

A.FM+0 PUD address of device to load task
A.FM+2 Address of STL node for this task, or zero
A.FM+4 UIC for task to run, or zero if not specified
A.FM+6 ATL address of task which requested this one, if requested by EXEC$ or FIX$

2 Waiting or stopped for single group of event flags (states WF0, WF1, WF2, WF3, ST0, ST1, ST2, ST3):

A.FM+0 Mask for flags being waited for in relevant event flag word (A.EF+0, A.EF+2, .COMEF, .COMEF+2)

3 Waiting or stopped for all groups of event flags (states WF4, ST4):

A.FM+0 Mask for flags 1-16 (A.EF+0)
A.FM+2 Mask for flags 17-32 (A.EF+2)
A.FM+4 Mask for flags 33-48 (.COMEF)
A.FM+6 Mask for flags 49-64 (.COMEF+2)

4 Waiting for Executive semaphore (state WSM):

A.FM+0 Mask for semaphore being waited for

5 After task exit (states EXT, STN):

A.FM+0 Reason for exit (LO byte), exit flags (HI byte):

Bit 8 (000400) set if TKTN required
Bit 9 (001000) set if I/O rundown required
Bit 10 (002000) set if task exited with valid status

A.FM+2 Task exit status

6 Waiting for directive (state WDI):

A.FM+6 Error status to return to task if directive fails

7 Directive complete (state DIF):

A.FM+6 Error status to return to task

8 Task marked for extension (MEX):

A.FM+0 Previous task size (from A.TZ)

3.4.14 Task's Run Partition (A.PD)

This is the TPD address of the partition in which the task is running. It is not necessarily the same as the partition in which the task was installed. See Section 3.7 for details about the TPD.

3.4.15 Swap Address (A.SA)

When a real-time task is checkpointed or a scheduler controlled task is swapped out of memory, the system records a swap address which defines the space allocated on the swap file to hold the checkpointed or swapped task. See Section 4.2.3 for further details about swapping.

3.4.16 Current Task Size (A.TZ)

This is the size of the task's impure area. It might differ from the task's initial size (S.TZ) if the task has issued any extend task (EXTK$) directives.

3.4.17 Swap I/O Count (A.SW)

This is the number of I/O requests currently in progress which the appropriate handler has freed for swapping. A task can be swapped or checkpointed if its I/O in progress count is equal to this count.

3.5 System Task Directory (STD)

The System Task Directory (STD) is a table that provides information about each task installed in the system. The information recorded in a task's STD entry includes the following:

- Information required when the task is not active (for example, receive queue listhead).
- Information required to activate a task (for example, task name, disk address, size of load image).

IAS tasks are referred to by name, and the STD can be searched for an indicated taskname. The STD is structured to enable this search to be performed rapidly, without imposing naming conventions, order of installation, or the declaration of a large memory area.

The STD consists of a fixed table of entry pointers (alpha table) for the maximum number of installed tasks (specified during system generation) and a 16-word entry for each task that is installed. The table is maintained by the programs that install and remove tasks such that the number of entries is known and consecutive table words point to STD entries ordered alphabetically by taskname. Thus, a taskname can be found rapidly using a binary search, and memory is not dedicated for STD entries until it is needed.

The 16-word block of memory for an STD is taken from the node pool when a task is installed and is returned when a task is removed.

3.5.1 Default Task Partition (S.TD)

This is the TPD address of the task's default partition, determined when the task was installed. The default partition will be the partition used to run the task if no partition is specified when the task was requested.

3.5.2 Flags Word (S.FW)

The flags word contains information relating to the status and characteristics attributed to a task. The flags word bits are set when the following actions occur:

- A single-user task is currently fixed in memory (SF.FX)
- A task is to be automatically removed (SF.RM)
- A task is disabled (SF.TD)
- A task is being fixed in memory (SF.BF)
- A task is to be removed on exit (SF.XT)
- A task is multiuser (SF.MU)
- A task is unable to receive data or references (SF.XS)
- A task is never to be aborted (SF.XA)
- A task is never to be disabled (SF.XD)
- A task is never to be fixed in memory (SF.XF)
- A task is never to be checkpointed (SF.XC)
- A task can receive from VSDR$ directive and be requested/resumed by SRFR$ directive tasks that do not have real-time directive privilege (SF.SR).

3.5.3 Default Priority (S.DP)

When a task is being built or installed, the user has the option of specifying a priority at which that task is to execute. If a run priority is omitted, the default priority (normally set at 50) will be used.

3.5.4 System Disk Indicator (S.DI)

When a task is to be loaded into memory for the first time, the system disk indicator is used to indicate which device is to receive a "load task image" request. The indicator provides a Physical Unit Directory (PUD) entry index (not an actual PUD address). For example, a zero indicates the request queue for the device unit represented by the first (entry zero) PUD entry. See Section 3.6 for details about the PUD.

3.5.5 Size of Load Image (S.LZ)

This is the size of the task (in 32-word blocks) which must be loaded when the task runs. It includes the task header and the root segment of an overlaid task, but does not include any overlays. See the *IAS Task Builder Reference Manual* for a full description of overlays.

3.5.6 Installed Task Size (S.TZ)

This is the amount of memory (in 32-word blocks) to allocate for the task when it is initially loaded. The amount of memory includes the task header, the root segment, overlays and the task extension. (In other words, it is the size of the contiguous area needed for the task itself, if any).

3.5.7 Active Versions Count (S.AV)

More than one version of a task can be active. This is a count of the number of active versions and is used for node accounting purposes (see Section 3.2.3).

3.5.8 Node Pool Utilization Limit (S.PV)

Each installed task has a node pool utilization limit which is used for node pool accounting purposes. See Section 3.2.1 for details about the node pool utilization limit.

3.5.9 Node Pool Usage Count (S.PU)

The node pool usage count is a count of nodes used by a task. See Section 3.2.2 for details about the node pool usage count.

3.5.10 Load Image First Block Number (S.DL)

The load image first logical block number indicates to the system the disk address from which a task is to be loaded.

3.5.11 GCD Node Address for Pure Area (S.PA)

When a task has a pure area, an entry for it is created in the Global Common Directory (GCD). If a task has such an area, its GCD node address is stored here. See Section 3.8 for details about the GCD.

3.6 Physical Unit Directory (PUD)

During System Generation, you specify which devices are to be used in the system. The Physical Unit Directory (PUD) is a fixed list of entries describing each device in the system. The PUD contains the following information:

1 The device name and unit number.

2 The type of device (for example, RK06, TU16).

3 The software priority at which device interrupts are to be serviced.

4 The External Page address of the device controller.

5 The device interrupt vector address.

3.6.1 Flags Byte (U.FB)

The system records the current state of a device and the operations occuring on that device by reserving a flags byte which sets bits when the following actions occur:

- A handler task is declared resident.
- The device can be used to install tasks (that is, the handler will recognize task load and record requests).

3.6.2 Device Independent Indicators (U.C1)

The system requires information regarding the characteristics of each device. During System Generation these characteristics are recorded in the PUD entry for the associated device. The device independent indicators set bit definitions when a device is:

- Record-oriented (for example, a card reader).
- Carriage-controlled (for example, a line printer).
- A teleprinter device (for example, an LA30).
- A directory device (for example, disk).
- A single-directory device.
- A sequential device (for example, a magnetic tape).
- An interactive IAS terminal.
- Allocated exclusively to one user.
- An input spooled device.
- An output spooled device.
- A pseudo device (that is, not a physical device and does not have a handler).
- A communications channel device.
- A FILES-11 device.
- A mountable device.

3.6.3 Device Dependent Indicators (U.C2/U.C3)

Some devices have characteristics specific to that type of device. These two words are used to record this information.

3.6.4 Size of Block, Buffer, Line (U.C4)

For a record-oriented device, this is the line length. For a random access device it is the logical block size (always 512. bytes).

3.6.5 Attach Flag (U.AF)

A task can obtain exclusive use of a device by "attaching" itself to that device. A word is reserved in each PUD entry that contains either of the following:

- The ATL address of the attached task.
- The PUD address of the owning device (if the device is an interactive terminal or an exclusive IAS device).

3.6.6 Redirect Pointer (U.RP)

I/O devices can be redirected from one device to another. A word is reserved in each PUD entry which contains either:

- The PUD address of the device which is to be redirected
- Its own PUD address, if the device is not redirected.

3.6.7 Handler Task ATL Node Address (U.HA)

When a handler is loaded and running, the ATL address for the handler is set in the PUD entry.

3.7 Task Partition Directory (TPD)

The Task Partition Directory (TPD) is a fixed list of entries describing each partition in a system. Such information includes:

- The name of the partition.
- The base address in memory of the partition.
- The size (in 32-word blocks) of the partition.
- The type of partition (user-controlled, system-controlled or timesharing).

When a task requests memory space in a system-controlled partition, the system needs to determine whether sufficient contiguous space is available to enable the task (and any SGAs) to be loaded. Therefore, the system records information on *holes* (or free memory space) within a partition by maintaining "hole pointer addresses." If insufficient memory is available for a task, the hole pointer is indicated by a zero. Similarly, the last free area in a partition is signified by a zero pointer.

For a timesharing partition, memory allocation is controlled using a Memory Usage List (See Section 3.17).

3.8 Global Common Directory (GCD)

The Global Common Directory (GCD) contains information regarding the following:

- SGAs (shareable libraries and common areas)
- Task read-only areas
- Dynamically created regions

- Task read/write regions (that is, impure resident overlays and VSECTs). See the *IAS Task Builder Reference Manual* for a description of virtual program section (VSECTs).

The GCD is a doubly-linked list that contains one node for each of the above items. The following fields, described in Sections 3.8.1 to 3.8.3, are of particular interest to the real-time programmer.

3.8.1 SGA Status (G.GS)

The status field is used to control the loading and recording of global areas. The status field might have one of the following values:

- SGA not in use (GS.NUL)
- Load request queued (GS.LRQ)
- Load request successful, global area resident in memory (GS.LRS)
- Load request failed (GS.LRF)
- Record request queued (GS.RRQ)
- Record request successful (GS.RRS)
- Record request failed (GS.RRF)

3.8.2 Active Reference Count (G.AC)

An SGA is only loaded into memory when required by a task. When a task exits or is checkpointed or swapped, that task no longer accesses any SGAs. Therefore, the system maintains a count of the number of memory resident tasks which reference an SGA. If this count becomes zero, the SGA is no longer being accessed by any task and therefore the area can be removed from memory.

3.8.3 Installed Reference Count (G.IC)

A task can bind to an SGA at task build time or dynamically attach to a region via the ATTACH REGION (ATRG$) system directive. The installed reference count is incremented for each attached task or installed task which binds to an SGA. It is used to prevent an SGA or region being deleted or removed while there are still references to that region.

3.9 Input/Output Request Queue (IRQ)

When a task requests an I/O operation for a device (using the QIO$ directive), the system requires a mechanism for recording such a request. All requests for a device (all units) are in a single queue called the I/O request queue. Additionally, the Executive creates I/O requests to load or record a task image, or rundown I/O on an exiting task. I/O requests are placed in priority-ordered queues of requests known as the I/O Request Queue (IRQ).

3.10 Clock Queue (CKQ)

The system can schedule operations to be performed at some future time. When the scheduled operation "comes due," the system must perform the indicated operation. Therefore, a queue of scheduled operations, called the Clock Queue (CKQ), is maintained.

This queue contains one node for each operation scheduled to be performed at some future time. A schedule delta time (see Section 3.10.1) in the first node of the Clock Queue is decremented at each clock tick until the node comes due (that is, at delta time zero). At this point, the indicated operation is performed.

Clock queue nodes are linked in the order in which they will become due and the schedule delta time in each node (except the first) is relative to the due time of the previous clock queue node.

Two types of operations can be scheduled for future execution:

1 Mark time operations

2 Task scheduling operations

Consequently, each node within the Clock Queue is marked by its type. The clock queue is also used by the IAS scheduler to record the end of the current task's quantum.

3.10.1 Schedule Delta Time (C.SD)

The first entry in the clock queue (that is, the node due next) contains the clock queue countdown, scheduled from delta time zero (that is, the actual time that the request was queued). At each clock tick, this count is decremented until the first node comes due. The next node then contains the new clock queue count. In this way, only the topmost count is decremented because all entries are scheduled relative to one another.

Figure 3–4 shows examples of three types of clock queue operations:

1 An illustration of a clock tick becoming due.

2 An illustration of one clock node being deleted from the queue and another added.

3 An illustration of a node being inserted into the queue (thereby causing the delta time of the next node to be adjusted).

3.11 Asynchronous System Trap Queue (ASQ)

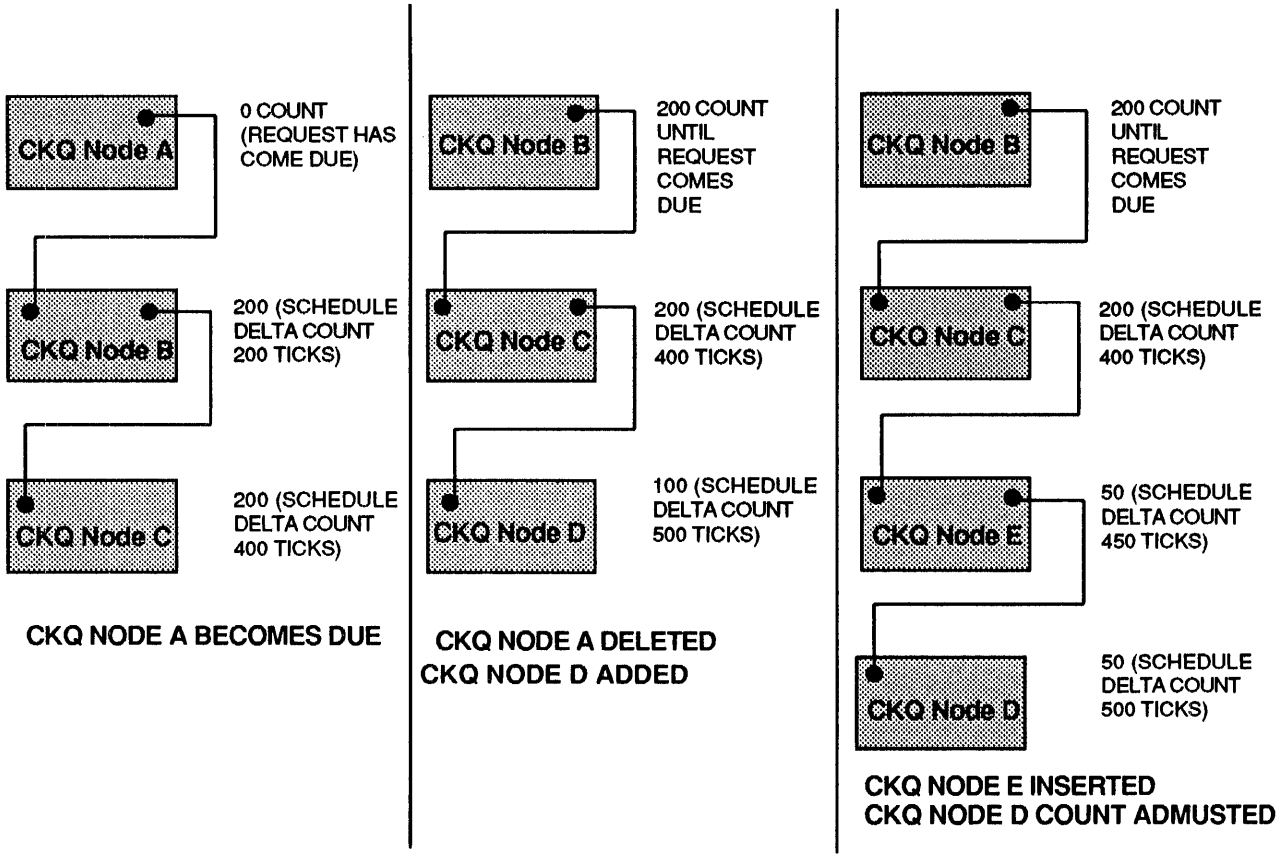
An Asynchronous System Trap Queue (ASQ) is a queue of asynchronous system trap notifications awaiting service from an active task. Such trap conditions are normally serviced immediately, unless a task is already servicing an AST or has inhibited AST recognition. When two or more notifications (ASQ nodes) are queued, they are serviced on a first-in/first-out (FIFO) basis.

The ASQ listhead is contained in the task's ATL node and is used when an AST has been signalled but has not been serviced by the task.

An AST condition might have additional parameters (depending on the reason for the AST) which will be pushed onto the task's stack in addition to the task's PS and PC (for example, a Mark Time AST has an associated event flag number). These parameters are contained in the AST node.

The task's ASQ is also used to contain completed I/O requests and completion indicators for spawned tasks. This is done so that the associated status is available at a time when the task is known to be in memory, when it is next run.

Figure 3–4 Clock Tick Recognition



3.12 Send/Receive Queue (SRQ)

A task can send data via the system directives SEND DATA (VSDA$/SDAT$) or SEND DATA AND RESUME OR REQUEST RECEIVER (VSDR$/SDRQ$). The SEND DATA directives queue a variable-length data block for a task to receive.

The SEND/RECEIVE Queue (SRQ) is a deque with a listhead in the STD entry for the receiving task. The queue contains one node for each block of data sent to the task defined by the STD entry. Note that the SRQ listhead is contained in the STD (and not the ATL) because data can be queued to a task which exists in the system (that is, a task which is installed) but is not active (that is, the task has no ATL entry).

3.13 Send/Receive By Reference Queue (RRQ)

The SEND/RECEIVE by Reference Queue (RRQ) is a single list that contains all the data packets for all references which have been sent using the SEND BY REFERENCE (SREF$ and SRFR$) directives, which have not yet been received (by RREF$). SCOM has a single listhead for this list. A single list is used, rather than a list per task as for the SRQ, to simplify scanning when a sending task exits.

3.14 Spawn Task List (STL)

The Spawn Task List (STL) contains one node for each task spawned as a result of a task having issued a SPAWN TASK (SPWN$) system directive. In addition, if a command line was issued with the directive, the node contains the command line until it is picked by the GET MCR COMMAND LINE (GMCR$) system directive.

To avoid a search of the STL to find the relevant node, a spawned task has a pointer in its task header to its STL node. The only purpose of the STL is to enable the Executive to find all tasks spawned by another task when it exits (so that the linkage can be undone).

The command line section exists only while a command line is actually present. The command line is deallocated when the spawned task performs a GET MCR COMMAND LINE directive. The command line section of an STL node is allocated before the remainder of the node. This avoids node pool fragmentation which would otherwise occur with certain system tasks.

3.15 User Task List (UTL)

The User Task List (UTL) is a deque of entries, each of which represents a timesharing scheduling queue level.

Each entry in the deque contains the listhead of a deque of nodes (actually ATL nodes) for tasks which are currently active in that scheduling level.

The scheduler can promote and demote tasks between levels on the basis of their activity history by unlinking nodes from one level and relinking them into another.

Jobs in the level 1 UTL entry get highest priority service from the scheduler. The maximum number of levels is normally specified at System Generation time. The lowest scheduling level can be specified as a batch scheduling level and reserved for batch tasks.

3.16 Swap File List (SFL)

The Swap File List (SFL) is a list of swap files currently available to the system. The SFL is used by the swap file allocation/deallocation routines in conjunction with the swap file bitmap, which indicates which swap file blocks are available. Additionally, the SFL is used when translating a swap file block number into a PUD address (device) and disk Logical Block Number (LBN). SFL entries are in ascending order of swap-file.

3.17 Memory Usage List (MUL)

A Memory Usage List (MUL) is used to control memory allocation in timesharing partitions. There is a separate MUL for each partition, with the listhead contained in the corresponding TPD entry.

MUL contains one entry for each memory segment currently allocated within the partition. Each node contains the following information:

1 The base address in physical memory of the segment.

2 The size of the segment in 32-word blocks.

3 The size of the free, unallocated area ("hole") above the segment.

4 The identity of the task, global area or region to which the memory segment is allocated.

3.18 Fixed Task List (FTL)

The Fixed Task List (FTL) is a deque of ATL nodes for tasks that have been fixed in memory but are not active. When a fixed task is made active, its FTL node is relinked into the ATL. When the task exits, its node is relinked into the FTL.

3.19 User Job Node (UJN)

A User Job Node (UJN) exists for every task under the control of the IAS scheduler that can be scheduled to run. The node contains all information required for resource management for the task. A job node is picked from the job node pool whenever a task is initiated and is returned to the pool when the task terminates.

The listhead for job nodes is contained in the terminal node for the terminal for which the task is running. Each job node is cross-linked with the corresponding ATL node if one exists for the job.

3.20 User Terminal Node (UTN)

A User Terminal Node (UTN) is allocated to every device that is specified as a timesharing terminal during system startup. Terminal nodes are chained into a list for the CLI currently servicing those devices. A terminal node can be linked to only one CLI node at any one time.

The terminal node contains the timesharing device characteristics and information about the current activities for the terminal (for example, the currently active CLI task.)

See the *IAS Guide to Writing Command Language Interpreters* for further details about CLIs.

3.21 Command Interpreter Table (CIT)

The Command Interpreter Table (CIT) contains an entry for each CLI in the system. The maximum number of concurrent CLIs is specified during System Generation, determining the number of CIT entries established. The linked list of UTNs which a CLI is currently serving is pointed to by that CLI's CIT entry.

3.22 Device Table (DVT)

The Device Table (DVT) supplements the information held in the Physical Unit Directory (PUD) in SCOM. The Device Table contains information about device usage for IAS timesharing users. See Section 3.6 for further details about the Physical Unit Directory.

3.23 Device Load Table (DLT)

The Device Load Table (DLT) contains one node for each device mounted in the system for timesharing users and is used by the Timesharing Control Primitives (TCP) for device management.

3.24 Job Node Pool (JNP)

The Job Node Pool (JNP) is a pool for currently unused job nodes. The number of nodes is specified during system generation.

3.25 Terminal Node Pool (TNP)

The Terminal Node Pool (TNP) is a pool for currently unused terminal nodes. The number of nodes is specified during System Generation.

3.26 Task Header Contents

The task header contains information about the current state of a task. Section 3.3.6 contains information about task header usage. This section describes the fields within the task header in more detail. Appendix C of the *IAS Task Builder Reference Manual* lists the complete contents of the task header.

3.26.1 Context Reference 1 (H.CR1)

If a task is built to include the floating point capability, this word contains a pointer to the floating point save area (see Section 3.26.21). H.CR1 is zero if the task does not include the floating point capability.

3.26.2 Mapping Registers

The page descriptor registers, page address registers, page flags registers, page length registers, and page offset registers all contain information about the task's memory mapping context. There is one of each of these registers for each APR (n=0 to 7). Some of this information is set up when the task is installed, and the Executive uses it when loading the task into memory. The Executive uses the page descriptor registers and page address registers to load the hardware memory management registers. The other registers describe the current software mappings of each APR for the task.

3.26.2.1 Page Descriptor Registers (H.PDn;n=0-7)

The Executive sets up the page descriptor registers in the copy of the task header in memory when it loads the task. Whenever the Executive activates the task it uses these page descriptor register contents to set up the hardware memory management page descriptor registers.

3.26.2.2 Page Address Registers (H.PAn;n=0-7)

The Page address registers describe the physical address mapping of the task's virtual address space. The Executive sets up the mapping registers in the header when the task is loaded and subsequently uses the registers to set up the hardware page address registers each time the task is made active.

Before the task is made active, the task header (in the task image file) contains the GCD address of any registers mapped by the task in the appropriate H.PAn. These GCD addresses are set up by INSTALL. During loading, the Executive uses these addresses to check whether the required regions are already in memory. If they are not, the Executive initiates their loading. Whenever the Executive activates the task, it changes any page address register that maps a region to contain the correct mapping address. This ensures that the page address registers are correct even if the regions have been moved in memory.

3.26.2.3 Page Flags Registers (H.PFn;n=0-7)

These flags describe the use and characteristics of the corresponding APR. They are set up by INSTALL and can be any of the following:

- PF.WIN The APR is the first of a window.
- PF.WN0 The APR is the first of window 0.
- PF.CON The APR is a continuation of the previous APR (the region needs more than one APR to map it).
- PF.RAC The region has been accessed.
- PF.MAP The APR is mapped on to a region (the corresponding H.PAn contains the GCD address)

3.26.2.4 Page Length Registers (H.PLn;=0-7)

The page length registers contain the total size of the window for the region mapped by the corresponding APR. Windows are described in the *IAS System Directives Reference Manual.* INSTALL sets up the page length registers for installed regions. For mapping changes made during execution of the task the Executive sets up the page length registers dynamically.

3.26.2.5 Page Offset Registers (H.POn;n=0-7)

When a region of a task is mapped by more than one APR, H.POn defines the start of the area mapped by the nth APR. Its value is the offset of the area from the start of the region.

3.26.3 Task's Registers, Program Status Word, Program Counter and Stack Pointer (H.TRn;n=0-5, H.TPS, H.TPC, H.TSP)

The Executive saves the task's registers, status word, program counter and stack pointer in the task header wherever the task ceases to be the currently executing task. The registers, program status word, PC and SP are restored from these words in the header when the task is made active again.

3.26.4 Task's Initial Program Status Word, Program Counter and Stack Pointer (H.IPS, H.IPC, H.ISP)

These words are set up by the task builder to give the initial task entry conditions. They are loaded into H.TPS, H.TPC and H.TSP when the task is requested.

3.26.5 Debugging SST Vector Table Address (H.DSV)

This address is used by the Executive to locate the debug SST trap vector table if one exists (see Section 2.4.2).

3.26.6 Task SST Vector Table Address (H.TSV)

The task builder sets this word to point to the task's internal SST vector table if one was specified using the task builder's TSKV option. See Section 2.4.2 and the *IAS Task Builder Reference Manual* for further details.

3.26.7 Default User Identification Code (H.DUI)

This UIC is set by INSTALL to indicate the UIC under which the Executive is to run the task if none is specified when the task is requested. In this event the Executive copies H.DUI to H.UIC when it activates the task.

3.26.8 User Identification Code (H.UIC)

H.UIC contains the UIC under which the task runs. The Executive sets up H.UIC when it loads the task for the first time. This UIC value is used during execution to check the task's access rights to files and regions.

3.26.9 Task Attributes (H.TAT)

This word indicates that the task has certain attributes that were specified when the task was built:

- HT.FRQ—If set, the task requires receive queues to be flushed on task exit.
- HT.NWD—If set, the Executive is not to not wait for nodes for this task.

3.26.10 Size of Read/Write Resident Overlay Region (H.RWZ)

The task builder sets up this word if the task has read/write resident overlays. The Executive uses this word when creating a GCD node for the region while first loading the task. INSTALL checks this word to determine whether the task has read/write overlays.

3.26.11 I/O Queue Listhead (H.IOQ)

These two words are the listhead of the deque of I/O request nodes which are waiting to be processed by the task. They are only used if the task is a device handler. A node is added to the deque by the QIO directive processing and is dequeued by the handler when it uses the ..DQRN or ..DQRE routines. The IAS Guide to Writing a Device Handler Task provides further information.

3.26.12 Task Flags (H.EAF)

These flags indicate that the Executive must perform certain actions for the task:

- HF.RMC—If set, indicates that MCR must be recalled when the task exits.
- HF.LPA—If set, indicates that the task's LUN assignments need to be completed (that is, for first time load).

3.26.13 Wait-for-nodes Retry Count (H.WNCT)

If "wait-for-nodes" was specified for the task when it was built, INSTALL sets H.WNCT to a system constant value. This causes the Executive to continue trying to get nodes for the task (for about 10 seconds) if, at any time during the task's processing, nodes are unobtainable.

3.26.14 Directive Privilege Flags (H.PVDI)

This byte contains flags that indicate which directives the task can issue.

- SF.RT—If clear, the task can issue real-time directive privileged directives.
- SF.PLS—If clear, the task can issue memory management directives.

These groups of directives are described in the *IAS System Directives Reference Manual*. Real-time tasks always have these flags clear.

3.26.15 Spawned Task Node Address (H.STLN)

If the task is a spawned task, this word points to the Spawned Task List (STL) node for this task. The Exective needs to locate the STL node when the spawned task exits so that it can notify the spawning task and return the STL node to the pool.

3.26.16 Pure Area Attachment Descriptor Block Address (H.PADB)

When it initializes the attachment descriptor blocks (ADBs) for a task, INSTALL sets this word to point to the ADB for the task's pure area.

The Executive uses this pointer when loading the task for the first time because it needs to set up the GCD address in the ADB and the appropriate page address register.

3.26.17 Header Check Word (H.CHK)

INSTALL sets this word to the 16 low-order bits of the logical block number of the task image on disk. The Executive tests this value each time the task is loaded into memory as a check that the header has not been corrupted.

3.26.18 Resident Overlay Region APR (H.RWAP)

The task builder sets this word to point to the page address register word (H.PAn) which the Executive is to use for loading the task's read/write resident overlay region, if any. When the Executive loads the task for the first time it sets the task header word H.PAn to the address of the GCD node representing the read/write resident overlay region. This results in the region being loaded from disk when the Executive loads the task.

3.26.19 Task's Maximum Extension (H.MEX)

When a task extends itself during execution, the extended area cannot exceed the value contained in H.MEX. This value is recorded in units of 32-word blocks. It is set by the task builder when the task is linked.

3.26.20 Logical Unit Table (H.LUT)

H.LUT marks the start of the task's logical unit table (LUT). The LUT identifies which device is assigned to which logical unit number (LUN). The first word contains the number of entries in the table. This is followed by the the appropriate number of two-word entries. Each entry has the following form:

- Word 0 PUD address of device to which LUN is assigned.
- Word 1 Open file information (used by ACP task).

3.26.21 Attachment Descriptor Blocks Area

Attachment descriptor blocks area follows the variable length LUT in the task header. H.ADB points to the area. Each two-word attachment descriptor block (ADB) has the following format:

- Word 1 Address of the GCD node for the region.
- Word 2 Flags:
  - RF.RED Task has read access to region.
  - RF.WRT Task has write access to region.
  - RF.EXT Task has extend access to region.
  - RF.DEL Task has delete access to region.
  - RF.XDT Task cannot detach from region (used, for example, for tasks' pure area ADB).
  - RF.ITA The attach was done during INSTALL (set for regions which have ADBs generated during task build).

ADBs for the task's pure area, regions linked to a task at task build time, and the two possible resident overlay regions (read only and read/write), are created automatically by the task builder. If the task dynamically attaches to other regions further ADB space is generated using the ATRG option of the task builder (see the *IAS Task Builder Reference Manual*).

INSTALL initializes the ADBs for the regions which are linked at task build time. However it cannot fully initialize the read/write resident overlay ADB (if any) because the required GCD node cannot be created until the corresponding copies of the task are loaded.

3.26.22 Floating Point Save Area

This area, pointed to by H.CR1, follows the ADBs in the task header. When a task ceases to be the currently active task the Executive checks H.CR1. If it is non-zero, the Executive saves the floating point context in this area. Similarly the Executive restores the floating point context from this area when it makes the task active again.

4 Memory Allocation and Scheduling

This chapter describes the following system functionality:

- Three types of IAS partitions
- Scheduling mechanism
- System checkpointing and swapping capabilities
- Use of fixed tasks
- Memory allocation

4.1 Partitions

Partitions are areas of contiguous locations in memory, declared during System Generation, that are used for task execution. Partitions provide a means of controlling memory allocation for task execution. A partition can contain executing tasks and tasks that are permanently resident in memory (fixed). IAS supports three types of partitions:

1 User-controlled partitions

2 System-controlled partitions

3 Timesharing partitions

The name, size, and type of each partition are specified during System Generation and cannot be changed without another System Generation. A default partition is also specified during System Generation. The default partition is the partition in which a real-time task executes if no other partition is defined when the task is built or installed. Scheduler-controlled tasks (that is, those tasks under the control of the IAS scheduler) always execute in the active Timesharing partition.

All memory that is not required for the IAS Executive and System Communications Area (SCOM) is available for partitions.

4.1.1 User-controlled Partitions

A user-controlled partition can contain only one task, shareable global area, or dynamic region at a time. The user has complete control over activity in the partition. User-controlled partitions are intended for the execution of real-time user tasks that need to be resident for long periods.

4.1.2 System-controlled Partitions

A system-controlled partition can contain one or more tasks at a time. The system controls the placement of tasks in memory. The only restriction is that a task cannot be loaded into a partition until there is a large enough contiguous memory area available within the partition for each loadable task segment. The Executive does not move task segments in a system-controlled partition, so fragmented free memory space cannot be collected together.

4.1.3 Timesharing Partitions

A timesharing partition in IAS is similar to a system-controlled partition. However, task areas resident in the partition can be moved in order to create larger areas of free memory to maximize the number of tasks that can be concurrently resident.

Tasks under the control of the IAS Scheduler will execute only in a timesharing partition. Real-time tasks can use any of the three types of partitions. There may be more than one timesharing partition in a system but only one is currently *active*. (That is, only one contains the tasks which are executing under scheduler control.) The active timesharing partition is specified during system startup. See the *IAS System Generation Guide* for further details.

When memory is required in a timesharing partition but there is no single area large enough, the Executive moves the allocated areas to the bottom of the partition to create the required area of contiguous memory locations. An area cannot be moved if an active I/O request specifies a buffer in that segment.

For most purposes, timesharing partitions are preferable to system-controlled partitions. System-controlled partitions are more suitable in the following circumstances:

1 When it is undesirable to move task areas in memory (for example, because a group of interacting executive privileged tasks are aware of each others' physical memory locations).

2 When the size of the system node pool is a limiting factor. Each allocated segment of a timesharing partition requires one node from the pool to describe the segment (a MUL node). System-controlled partitions do not require space from the node pool.

4.2 Scheduling Task Execution

This section describes the following functionality:

- How tasks are scheduled for execution.
- How the Executive copes when there is not enough memory to hold all active tasks.

4.2.1 Real-Time Task Scheduling

Every real-time task in IAS has an associated priority, in the range 1 to 250 (decimal). 250 is the highest, or most urgent priority. The Executive constantly tries to allocate the processor to the runnable task with the highest priority. When the currently running task ceases to be runnable (for example, because it waits for an event flag), the processor is allocated to the next-highest-priority runnable task.

Each active task in the system has an entry in the Active Task List (ATL). The entries are arranged in order of task priority. To find the highest priority runnable task the Executive simply scans this list starting with the highest priority active task, until it finds a task which is runnable (task state RUN).

The ATL is in many respects the most important data structure in the system, because of its fundamental role in task scheduling. The ATL is fully described in Section 3.4.

The ATL is scanned from the top when a significant event occurs. Thus, it is possible for a task to become runnable (for example, because an event flag for which it is waiting becomes set), but not to regain control of the processor from a lower priority task until a significant event occurs. This might occur, for example, if an event flag is set using the SET EVENT FLAG (SETF$) directive rather than DECLARE SIGNIFICANT EVENT (DECL$) directive.

Because of the structure of the ATL, tasks with the same numerical priority are in fact ordered. The ordering for such tasks is not defined and may change while the system is running.

The priority of a task is assigned when it is activated, and in general does not subsequently change. Note the following:

1 If a priority was specified in the directive or command that activated the task, that priority is used.

2 Otherwise, if a priority was specified when the task was installed, that priority is used.

3 Otherwise, if a priority was specified when the task was built, that priority is used.

4 Otherwise, the system default priority is used, as specified during System Generation.

To alter a task's priority while it is running, use the MCR ALT or PDS SET PRIORITY command, or from a task (including itself), use the ALTER PRIORITY (ALTP$) directive.

It is not possible to give definite rules about task priority allocation. However, Table 4–1 describes the general uses of priority ranges.

Table 4–1 Priority Ranges

Priority	Description
0	Used by the system null task.
1	Used by the timesharing null task. Should not normally be used by real-time tasks.
2–99	Available for user tasks. Tasks under the control of the IAS scheduler run at a priority of 100, so tasks in this priority band will run as background tasks, below timesharing activity. **NOTE:** In the active timesharing partition, a task requested at a priority less than the TSS2 scheduler control node are run under IAS scheduler control. See Section 4.2.2 for discussion of scheduler control nodes.
100–110	Used by the IAS scheduler and related system control tasks. Should not normally be used by user tasks.
110–220	This is the normal priority band for real-time tasks.
220–225	Used by system control tasks. Should not normally be used by user tasks.
225–239	Available for highly critical real-time tasks. If a task in this priority range loops, it will not be possible to abort it because it will not allow any of the control tasks to access the processor. Thus this range should only be used for fully tested programs.
240–247	Used by system tasks. Should not normally be used by user tasks.
248	Normal priority for device handler tasks.
249	Available for real-time tasks which must have priority above device handlers.
250	Used only for system disk handler.

4.2.2 Effect of the IAS Scheduler

This section describes how the IAS scheduler interacts with real-time task scheduling, and in particular how the scheduler uses the ATL.

If the system contains the IAS scheduler, any task that is requested in the active timesharing partition at a priority of 100 or less runs under the control of the scheduler.

The scheduler selects a scheduler-controlled task to run using the heuristics described in the *IAS System Management Guide*. The scheduler uses its own data structures for this purpose, in particular the User Task List (UTL).

When the scheduler has selected a scheduler controlled task to run, its ATL node is inserted into the ATL at the appropriate priority (normally 100). This causes it to be seen by the executive and to be run in the same way as a real-time task.

All other tasks are also linked into the ATL, but at a priority of 1. They are not be seen by the executive when scanning the ATL, because they are linked after the ATL entry for the timesharing null task, which is always effectively runnable. The scheduler only selects tasks that are able to run, thus the situation could occur where all scheduler controlled tasks are linked at priority 1. In this case, the ATL scan proceeds past the timesharing priority and services any lower priority real-time tasks.

Some scheduler controlled tasks are run at a higher priority than normal. These include the CLI for the console terminal (SCI) and system control tasks initiated from that terminal. These tasks normally run at priority 220; however, the principle of operation is the same.

The scheduler maintains four special control nodes in the ATL in addition to the ATL nodes for the scheduler controlled tasks. Although these nodes do not appear in the active task list (produced by ACT or SHOW TASKS/ACTIVE), they are linked into the ATL in the normal manner.

When a scheduler-controlled task is selected to run, the Executive inserts the task's ATL node in the ATL between two control nodes. If it is a normal scheduler controlled task, the node immediately above it is TSS1 and the one below is TSS2. The task runs at the priority of the TSS1 node (100). If the task is a high priority scheduler controlled task then the task's ATL node is linked in the ATL immediately below a high priority equivalent of TSS1, called THATL. The TSS2 node is re-linked below the scheduler controlled task's ATL node. The task runs at the priority of the THATL node (220) in this case. TSS1 and TSS2 have states TS1 and TS2 respectively. THATL has state TS1 when a high priority scheduler controlled task is to be executed or is executing and state SUS if not.

4.2.3 Fitting Active Tasks into Memory

It might not be possible to fit all currently active tasks into available memory at the same time. In this case, the Executive makes the best use of the memory using the following basic scheme:

1. Real-time tasks with priorities greater than scheduler controlled tasks are fitted into memory in priority order, that is, high priority tasks are given memory in preference to lower priority tasks.
2. If there is still memory available, it is shared between all tasks under the control of the IAS scheduler, so that each task is in memory for some of the time.
3. If memory is still available, it is allocated to real-time tasks with priorities lower than the TSS2 control node (see Section 4.2.2). This too is done in strict priority order.

An active task might have to be temporarily removed from memory to make room for other tasks. This process is called checkpointing for a real-time task, and swapping for a scheduler controlled task. Different names are used because of the different way the system decides to remove tasks from memory.

When tasks are removed from memory by swapping or checkpointing, they are temporarily stored on swap files created on disk. These are created using the SWAP (MCR) or CREATE/SWAPFILE (PDS) command.

Sections 4.2.4 and 4.2.5 describe further the operation of checkpointing and swapping.

4.2.4 Checkpointing

Checkpointing is a mechanism to aid real-time tasks to obtain memory to execute as soon as possible after they are requested. If sufficient memory is already available when a task is requested, the task will be loaded. If memory is not available, the system will checkpoint out of memory sufficient executing task(s) of lower priority than the requesting task to make the desired space available. The system can only checkpoint tasks which have the attribute of being checkpointable.

When the Executive tries to run a real-time task, it attempts to make room in memory using the following procedures:

1 The Executive first tries to find any stopped tasks that can be checkpointed. It checkpoints any stopped tasks one by one until there is sufficient space in memory or there are no more stopped tasks. Section 4.2.4.3 describes stopped tasks.

2 The Executive then tries to checkpoint lower priority real-time tasks to make room for the higher priority task. Section 4.2.4.1 describes this process.

3 If there is still insufficient space in memory the Executive looks for SGAs, regions, or tasks' pure areas which are not currently in use. It checkpoints these one by one until sufficient space exists or they have all been checkpointed.

If the partition is a timesharing partition, space can be made available for the high priority task by moving the task areas in the partition. The Executive moves the tasks if, at any time during the above procedures, there is sufficient memory for the high priority task but the free memory areas are not contiguous. Section 4.1.3 describes timesharing partitions.

If no room can be found in memory, the task is left waiting for memory (task state MRL; see Table 3–1).

4.2.4.1 Checkpointing Low Priority Tasks

- The Executive scans the ATL, starting with the lowest priority task, looking for a task that can be checkpointed. When the Executive finds such a task, it attempts to allocate space in the checkpoint file and, if successful, queues a record request to write the task's impure area to the swap file. When the task area has been written, the memory it occupied is released and the Executive tries again to find room for the highest priority task. If it fails, the entire process is repeated.

When a task has been checkpointed, its state is also set to MRL and, when it becomes the highest priority task waiting for memory (usually because the original task has been successfully loaded), the same process is repeated on its behalf.

The Executive only tries to find room for the highest priority task. In this context, consider the following situation. Task A has a size of 12K and a priority of 150. Task B has a size of 8K and a priority of 140. Both tasks are waiting for memory (task state MRL). After checkpointing lower priority tasks, 10K is available. Task B will not be loaded into memory, even though it could be fitted, because a higher priority task is waiting for memory.

A low priority task will fail to be checkpointed in favor of a higher priority task in the following situations:

1 Insufficient swap file space is available to hold the task. This should not arise if the system is used correctly

2 The task is not checkpointable.

3 The task has disabled checkpointing using the DISABLE CHECKPOINTING (DSCP$) directive.

4 The task has been fixed in memory (see Section 4.3).

5 The task has an I/O operation in progress, and has not been released for checkpointing by the handler. In this case, the task is placed into the state SFC (suspended for checkpointing). No further I/O requests will be dequeued and when all current I/O has been completed the task will be checkpointed.

6 The task (usually a handler) has connected to an interrupt vector.

4.2.4.2 Checkpointing SGAs, Regions, and Task Pure Areas

When a task has been checkpointed, the access counts are decremented for all SGAs and regions to which it is mapped, and for its pure area. If this count becomes zero (that is, if no other memory-resident tasks are mapped to it), the SGA (or region or pure area) is eligible to be removed from memory. However, the Executive will not checkpoint SGAs, regions, or pure areas if the required memory area can be obtained by any other means. This avoids unnecessary checkpointing since the SGAs, regions, and pure areas will be needed when the task(s) which uses them is returned to memory.

4.2.4.3 Stopped Tasks

A task might be waiting for an event to occur (for example, for an event flag to be set), and want to run at a high priority when it does occur, but not need to be in memory while it is waiting. For example, a process control task may be waiting for an external event which will be signalled by the setting of an event flag. There is no need for the task to be in memory while it is waiting, but when the event occurs, the task must run at a priority greater than scheduler controlled tasks.

A task can relinquish memory in this way by using the stop facility. A task that is stopped has indicated that it does not need to be resident in memory. When the Executive needs memory, before scanning the ATL, it removes from memory all stopped checkpointable tasks, irrespective of their priority. When a task ceases to be stopped, it will be reloaded according to its priority using the checkpointing algorithm described above. A task can stop itself in the following ways:

- By issuing a STOP (STOP$) directive.
- By issuing a RECEIVE DATA OR STOP (RCST$ or VRCT$) directive, when there is no data to be received.
- By issuing a RECEIVE BY REFERENCE directive (RREF$) with the stop option (WS.RST set in window descriptor block), when there are no references to receive.

In the above cases, the task will be unstopped and become runnable again if one of the UNSTOP (USTP$), SEND DATA AND RESUME OR REQUEST (SDRQ$ VSDR$), SEND BY REFERENCE AND RESUME OR REQUEST (SRFR$) or RESUME OR UNSTOP (RSUS$) directives is issued, specifying its task name.

A task can also stop itself in the following ways:

- By issuing a STOP FOR SINGLE EVENT FLAG (STSE$) directive.
- By issuing a STOP FOR LOGICAL OR OF EVENT FLAGS (STLO$) directive.

In these cases, the task will be unstopped and become runnable if the appropriate event flag (or one of the flags for STLO$) is set.

A stopped task can have an AST queued. In that case, it is temporarily unstopped (and hence reloaded into memory if necessary) while the AST is serviced. When AST service is completed, the task will again be stopped, unless it has been unstopped while the AST was being serviced (possibly by the AST service routine).

An AST service routine can stop itself in the same way as the main task. In this case the task will be treated like any other stopped task. However, no other ASTs will be serviced because the task is already in an AST service routine.

4.2.5 Swapping

Swapping is controlled by the IAS scheduler. Unlike checkpointing, which is strictly controlled by priority and hence is under the control of the real-time programmer, swapping is guided by two considerations:

1 To maximize use of the processor and memory.

2 To provide a fair service to all tasks under IAS scheduler control.

The scheduler uses a set of heuristics to determine the best task to swap out and the best task to load at any time. These are based on previous task behaviour and current activity. See the *IAS System Management Guide* for further details.

All tasks under scheduler control can be swapped; building the task non-checkpointable or issuing the DISABLE CHECKPOINTING (DSCP$) directive has no effect. However, a scheduler controlled task can use the stop facility to indicate to the scheduler that it does not need to be resident at that time.

When scheduler-controlled and real-time tasks are running in the same partition, checkpointing and swapping interact as described in Section 4.2.3. Scheduler-controlled tasks are still swapped by the IAS scheduler, using its heuristics, even when the memory is required for a real-time task.

Swap file space for a scheduler-controlled task is allocated when the task is activated, and is not released until the task terminates. Thus, the scheduler can never fail to swap a task because of lack of swap space. If space cannot be found when the task is activated, it will not be executed.

4.3 Fixing Tasks

A fixed task is permanently resident in memory, even when it is inactive. It can therefore be executed very quickly, because it does not have to be loaded into memory. Typical occasions when you might use a fixed task are as follows:

1 When a task must be executed very quickly (for example, because it controls a critical real-time process).

2 When a task is executed very frequently for very short periods.

4.3.1 Operation of Fixed Tasks

To make a task fixed, follow these procedures:

1 Build it with the attributes *fixable* (/FIXABLE or /FX) and *non-checkpointable* (/NOCHECKPOINT or /-CP).

2 Install it.

3 Make it fixed, either with the FIX command or by using the FIX$ system directive from within another task.

When it is fixed by a command or a system directive, the task is loaded into memory together with any SGAs to which it is bound, its read-only region, and its resident overlays. If memory is insufficient to load all segments without swapping or checkpointing any other tasks currently resident, the attempt to fix the task will fail. When the task is requested (by a REQUEST, SEND AND REQUEST, SEND BY REFERENCE AND REQUEST, or other directive), it is run immediately. This saves time for two reasons:

1 No time is spent accessing the disk to load the task.

2 There is no need to make memory available by swapping or checkpointing tasks currently resident.

The task then executes normally. In most respects, there is no difference between a fixed task and any other. (A fixed task can, for example, be overlaid.) However, Section 4.3.2 gives some important considerations regarding fixed tasks.

When the task terminates for any reason, the memory that the task occupies is not released. Regions to which the task were bound at task build time also remain in memory. Thus, the execution of the task may be restarted next time the task is requested, without having to reload the task.

If a multiuser task is fixed, it is associated with a particular terminal (TI). This is either the TI of the task that issued the FIX$ directive, or, if the FIX command was used, it is the terminal where the command was issued. It is also possible, in the FIX command, to specify an alternative TI which overrides the original assignment. When the task is requested, the fixed version is only run if its TI matches that of the request. Otherwise, a new version of the task is loaded, which operates in the usual way and releases memory when the task exits.

Although a fixed task remains in memory, it is important to note that it does not necessarily occupy the same locations in physical memory. It may be 'shuffled' by the Executive, to make the best use of memory if it is in a timesharing partition.

The memory occupied by a fixed task can be released by unfixing the task, either by using the UNFIX command or by using the UFIX$ system directive. If a task is unfixed while it is active, the memory it occupies is released when the task exits.

4.3.2 Special Considerations for Fixed Tasks

The essence of fixed tasks is that they are not reloaded every time they are executed. Thus, all storage locations altered during one invocation will retain their new value for the next invocation. The Executive performs no initialization on behalf of a fixed task, other than to reset the program counter (PC) to the task start address and the stack pointer (SP) to its initial value. The other six general registers retain the value that they had when the task exited. Thus, it is essential that any locations where initial values are important (for example, counters and flags) be reset by initialization code. One implication of this is that it is possible for a fixed task to retain information

from one invocation to the next. This could be used, for example, to maintain an error count in a communications task.

Information that the Executive maintains in a task's header is also retained from one invocation to the next. In particular, Logical Unit Number (LUN) assignments are not reset to their initial values. Where the initial assignment of a LUN is important, but can subsequently be changed by the task, the ASSIGN LUN (ALUN$) directive should be used to reset the assignment during initialization. See Section 6.1 for a full description of LUNs.

All other clean-up operations normally performed by the Executive upon task exit are performed in the usual way. In particular:

- The task's receive queue and receive-by-reference queue are flushed (unless the task has the /NOFLUSH_RECEIVE_QUEUES attribute).
- I/O rundown is performed for the task.
- All open files are deaccessed, and locked if appropriate.
- All dynamically attached regions are detached.
- All dynamically created address windows are unmapped and eliminated. Window mappings are not restored to their initial state except when a privileged task has dynamically remapped address windows which initially pointed to SCOM or the external page. In this case the initial window mappings are restored. Any SGAs that were linked to the task when it was built, and that have been dynamically unmapped, remain unmapped when the task commences execution again.

4.4 Memory Protection

The memory areas assigned to a task are protected from other tasks executing in the system. The memory allocated to a task is accessed via memory management hardware, as described in Chapter 1. This hardware ensures that a task can access only the memory which has been allocated to that task. The type of access, read-only or read/write, is also controlled by the hardware.

The portions of memory used by more than one task, such as shared libraries and common areas, are also protected. Common areas can be pure (read-only) or impure (read/write) and the appropriate access is enforced by the memory management hardware. Care must be taken to prevent two tasks from simultaneously modifying the same data when a common area is shared between tasks with read/write access.

Common areas and libraries are referred to collectively in IAS as shareable global areas (SGAs). (See Chapter 5 for a description of SGAs.)

5 Shareable Global Areas

This chapter describes shareable global areas (SGAs), including the three types of available SGAs and the characteristics the user can assign to them.

SGAs are independent units of code and/or data that can be accessed by more than one task concurrently.

You use the task builder to create SGAs. Refer to the *IAS Task Builder Reference Manual* for details and examples of how to build an SGA. An SGA must be installed before any task that uses the area can gain access. Refer to the *IAS PDS User's Guide* or the *IAS MCR User's Guide* for details of how to install an SGA.

SGAs are normally used for two reasons:

1 To reduce the memory requirements of a system by enabling several tasks to share a single copy of any commonly used code or read-only data areas. For example, the library SYSRES contains routines used by many system and user tasks, such as the File Control Services routines.

2 To allow several tasks to access and hence communicate through a common read/write data area. Section 2.5.2 illustrates the use of SGAs for intertask communication.

5.1 Installed Reference and Active Reference Counts

The executive uses an installed reference count to count the number of installed tasks that reference the SGA. An SGA cannot be removed from the system until all tasks that refer to the SGA have been removed (that is, the installed reference count is zero).

The executive uses an active reference count to count the number of resident tasks using an SGA. When all active tasks that access an SGA are removed from memory, the executive frees the memory allocated to the SGA.

5.2 Types of Shareable Global Area

The three types of SGA are as follows:

1 Resident libraries

2 Common areas

3 Installed regions.

These types differ in the way the Executive treats them during swapping (or checkpointing) and when all accessing tasks exit. The user chooses the type of SGA most suitable for the application.

Resident libraries are treated as follows:

- The Executive always loads a resident library into memory from its task image file.

- Whenever all tasks that are using the resident library are no longer in memory (either swapped/checkpointed out or exited), the Executive frees the memory area used by the resident library. No copy of the memory area is made. Thus, any changes that might have been made to the resident library disappear. The resident library is not suitable as a read/write area if the accessing tasks are liable to be swapped/checkpointed by the Executive during their operation.

Common areas are treated as follows:

- The Executive always loads a common area into memory from its task image file.
- Whenever all tasks that are using the common area are no longer in memory (either swapped/checkpointed out or exited) the Executive writes the memory version of the common area back into its task image file on disk. Thus, any changes that tasks make to a common area are permanent, even when the IAS system is closed down and rebooted.

Installed regions are treated as follows:

- When the first task to use an installed region becomes active, the Executive loads it from its task image file.
- Whenever all tasks that are using the installed region are no longer in memory (either swapped/checkpointed out or exited) the Executive writes the memory version of the installed region to the swap area on disk.
- When subsequent tasks that use an installed region become active, the Executive loads it from the appropriate swap file.
- Thus, the installed region's task image file remains untouched during an IAS system session, although any changes to the contents of the region are permanent throughout that session. The original (unchanged) region is regained when the IAS system is re-booted.

5.3 Accessing a Shareable Global Area

For a task to access an SGA, the following conditions must be satisfied:

1 The user must specify the name of the SGA when building the task. This is done byusing the SGA or RESSGA Task Builder option as described in the *IAS Task Builder Reference Manual*. The SGA must have already been built and its task image file and symbol table file (for example, SYSRES.TSK and SYSRES.STB) must be available. This is because the Task Builder resolves any references to the symbols in the SGA as well as reserving virtual address space for it.

2 The SGA must be installed.

3 The UIC under which the task runs must permit access to the SGA. An SGA can have its access permissions set when it is installed. The *IAS PDS User's Guide* and *IAS MCR User's Guide* describe the installing of SGAs.

5.4 Position-Independent and Absolute Shareable Global Areas

A shareable global area can be position-independent or absolute. Position-independent SGAs can be placed anywhere in the task's virtual address space. Absolute areas must occupy fixed addresses in the task's virtual address space.

The *IAS Task Builder Reference Manual* discusses position-independent and absolute shareable areas in detail.

5.5 Shareable Global Areas with Resident Overlays

On a system with the memory management directives, it is possible for an SGA to be overlaid, using the resident overlay technique. The facility is fully described in the *IAS Task Builder Reference Manual*.

5.6 Installation and Removal

SGAs are installed and removed from the system in a manner similar to task installation. See the *IAS PDS User's Guide* or *IAS MCR User's Guide* for a description of the INSTALL and REMOVE commands.

When a task that refers to an SGA is requested, the Executive loads the task and checks the status of the SGAs required for the task. If an SGA is not already loaded (that is, no other active task refers to it), the Executive loads the SGA, resorting to checkpointing and swapping as needed.

6 Input/Output Facilities

This chapter describes the input/output facilities available in the system by describing the use of logical units, Queue I/O system directives and spooling. It also describes the process of I/O rundown performed by the Executive when a task exits.

Device independence is maintained in IAS by the use of logical units. The programmer codes a task that reads from and/or writes to logical units rather than physical devices. Each logical unit referred to must eventually be associated with a physical device. The association can differ for different executions of the program. The logical units are referred to by a Logical Unit Number (LUN). The logical/physical device relationship is specified by the assignment of the LUN to the required file or device. See the *IAS System Management Guide* for further details about LUNs.

6.1 Device Assignments

Logical Unit Numbers have no connection with physical devices until the programmer makes device assignments for a particular task. Device assignments, for example, tell the system that LUN 1 for user task TNAME could be associated with DECtape unit 2. In this case, I/O operations for logical unit 1 are performed on DECtape unit 2 when the task executes.

The system resolves the correspondence between physical devices and LUNs by means of a Logical Unit Table (LUT). The LUT is contained in the task's header. The header contains a specified number of slots, each of which corresponds to a LUN. Each slot contains a pointer to the physical device last assigned to that LUN and a pointer to information about the open file (if any) on that LUN. Each time a task issues a QIO directive (see Section 6.3) the system locates the physical device required by indexing into the LUT, by the argument given as the LUN.

Each user task has its own set of logical unit assignments that can be created or altered in the following ways:

1 Using the PDS ASSIGN or MCR REASSIGN command to assign or reassign a LUN of a particular task from one physical device unit to another (see the *IAS PDS User's Guide* or the *IAS MCR User's Guide.*

2 Using the ASG (Device Assignment) option when using the Task Builder to build a task. This option declares the physical device unit that is assigned to one or more logical unit (see the *IAS Task Builder Reference Manual*).

3 Using the ASSIGN LUN (ALUN$) system directive at run time to assign a LUN to a physical device unit (see the *IAS System Directives Reference Manual*).

In the first two cases, the user task is unaware of the physical devices that are accessed through its logical units. The task issues a QIO directive, specifying appropriate LUNs, while the actual I/O takes place interchangeably on a wide variety of system peripherals. In the third case, the user task is aware of its device assignments, but unaware of any redirection done to the device.

When building a task, the user must specify the highest logical unit number (if greater than 6, the default) that is used in the task. For example, if logical unit numbers 1, 3 and 9 are used, the maximum units is 9. Table 6–1 lists default LUN assignments for the first six LUNs.

Table 6–1 Default LUN Assignments

LUN	Device
1–4	SY0 (user default device)
5	TI0 (user terminal device)
6	CL0 (default system output device)

You can define logical devices and logical units to maintain device independence in commands and programs. A similar mechanism is necessary on a system-wide basis so the system can continue to function when a particular device malfunctions and devices can be changed to permit more efficient use of the system. A pseudo device name is a name that the System Manager can associate with a physical device name. The pseudo device name has the same meaning to all users of the system. For example, the system operator's output device is referred to by all users as CO: (console output). The pseudo device name allows programs to write messages on the operator terminal without knowing which physical device is being used. The pseudo device names used by IAS are as follows:

SY—User default device

TI—User Terminal input

TO—User Terminal output

CI—System Console input

CO—System Consle output

CL—System Console log (normally the line printer)

MO—Message output

PI—Primitive interface

SP—Spool input/output disk and system work file disk

LB—System library disk

WK—Fast system workfile disk

NL—Null device

See the *IAS System Management Guide* for a more detailed description of pdeudo device names.

6.2 Device Handler Tasks

Device handler tasks control I/O devices. These tasks are similar to normal tasks within the system but with the following additional features:

- They usually contain an interrupt service routine to respond to hardware interrupts.
- They are allowed to gain access to any memory areas, including privileged areas.

By convention, device handler task names consist of two alphabetic characters, followed by four dots. For example, the line printer handler is named:

```
LP....
```

The appropriate handler must be resident before a device can be used. Device handlers can be dynamically loaded into memory and unloaded from memory in order to conserve space.

6.3 QIO System Directives

User tasks not using the File System can make I/O requests to device handlers by issuing QIO directives (see the *IAS System Directives Reference Manual*).

The Executive's main function in I/O operations is to handle I/O requests from tasks and pass the requests to the appropriate device handler task. The general method is as follows:

1 A QIO directive is issued by a task. The task specifies a number of parameters that are required in processing the I/O request. One of these parameters is the LUN.

2 The Executive examines the LUN parameter of the QIO directive to determine which device handler is to process the request. The particular device handler is chosen by mapping the LUN of a particular task into an entry in the physical unit directory, using the logical unit table.

3 The I/O request is put in the request queue of the appropriate device handler.

As explained below, the requesting task can either suspend operation until the I/O request is completed or continue to operate until interrupted by an asynchronous system trap (see Chapter 2). IAS permits parallel I/O requests to be issued by the same task. That is, the task continues executing after issuing a QIO; subsequently the task can issue further QIO requests without waiting for the previous request to be completed.

Some device handlers operate in conjunction with the File Control Primitives (FCP) to manipulate files. When an FCP routine is required, the device handler issues a SEND/REQUEST which initiates operation of the specified FCP routine.

I/O requests are queued by priority at requestor task priority unless otherwise specified. The handler tasks pick requests from the top of the request queue. Thus, preferential service is given to high priority requests. However, when appropriate, devices can be attached to a task, in which case only requests from the attached task or express request are dequeued. This continues until a detach-unit-from-task request is dequeued, causing requests to be dequeued by priority from the top of the I/O request queue once again.

The ability to attach and detach devices is controlled by access privileges defined for each device. Requests to attach a device are rejected if the requestor does not have the proper access rights. Because device handler tasks can service many units, they are not themselves attached.

I/O requests are queued only if the data passed with the directive (in the DPB) contains the correct arguments. After the device handler completes an I/O request, one or more of the following actions are performed for the user task (depending on the arguments in the DPB):

1 An event is declared and a specified event flag set. These functions allow the user program to perform synchronous I/O operations in the following manner:

   a. Issue a QIO directive specifying an event flag (this clears the event flag).

   b. Optionally, execute some code within the user program.

   c. Issue a WAITFOR directive specifying the same event flag.

2 The current user task status is saved, an Asynchronous System Trap (AST) is declared and the user task is started at the AST address specified in the DPB. These functions allow the user program to perform asynchronous I/O operations in the following manner:

   a. Issue a QIO directive specifying the starting address of the AST service routine within the user task.

   b. Execute other instructions (including further QIO directives if desired).

   c. Execute its AST code, unseen by the user's normal code, when the I/O is completed.

3 The status of the I/O operation is returned from the device handler to a 2-word user status buffer defined in the DPB.

Device assignments and QIO directives are further described in the *IAS Device Handlers Reference Manual.*

6.4 Spooling

IAS provides both automatic output spooling and input spooling for serial I/O devices.

6.4.1 Automatic Output Spooling

Automatic output spooling allows a task to perform output to a record-oriented device (for example, a line printer), as though it had exclusive access to the device. The system places all output directed to the device into a file on disk, and then when the output file is closed it enters the file created into a queue of output for the device. The file will be printed at some time in the future when it comes to the head of the queue.

The facility enables several tasks to direct output to the same spooled device at the same time, without their output being interspersed.

For automatic spooling to work, output must be done using the file system (FCS is described in the *IAS/RSX-11 I/O Operations Reference Manual*, and RMS-11 is described in the *Introduction to RMS-11*). It is not possible to perform output to a spooled device using the Queue I/O directive. Any attempt to do this will fail with a status of IE.PRI (privilege violation).

It is also possible to queue files directly to a spooled device, using the PDS PRINT and QUEUE commands, the MCR QUE command, or the PRINT$ macro from within a task. See the *IAS PDS User's Guide* or the *IAS MCR User's Guide* for details about these commands. See the *IAS/RSX-11 I/O Operations Reference Manual* for details about the PRINT$ macro.

6.4.2 Input Spooling

Input spooling is used for batch input for job files that are submitted from one or more card readers. Once installed, the input spooler is a multi-user task, called only by a card reader handler. More than one card reader can call the input spooler simultaneously.

Spooled files are temporarily stored in a file under UIC [1,4] on the SP device. A batch queue entry is made for each job file encountered on any input device.

6.5 I/O Rundown

A correctly written task should ensure that before exiting:

1 All its I/O operations have completed

2 All files have been closed

3 All attached devices have been detached.

However, it is possible for a task to exit without having completed the above actions. In this case, it is not possible to perform task exit processing in the normal way because:

1 Active I/O requests reference physical memory, which is deallocated when a task exits.

2 All I/O requests contain references to the task's ATL node, which is also deallocated.

3 Open files must be closed for the file system to operate correctly.

4 If a device was attached to an exited task, no other task will be able to access the device again until the system is rebooted.

For this reason, the Executive cleans up all I/O related items, if any, at the beginning of task exit processing. This process is called I/O rundown.

The Executive performs I/O rundown processing if and only if the task's I/O pending count is non-zero when it exits (or is aborted). As explained Section 3.4.3, this count is incremented for open files and attached devices, as well as for pending I/O requests.

Figure 6–1 shows the flow of I/O rundown processing in the Executive, in a simplified form. Note that:

1 Only one task at a time can have I/O rundown performed. If a task requires I/O rundown while it is being performed for another task, it is left in state 'IR1' (I/O rundown pending) until the other task completes I/O rundown.

2 I/O rundown depends upon the co-operation of every device handler in the system. A rundown request for a task is sent to each handler in turn (and will be sent to each handler once for each unit served by the handler). It is up to the handler to perform the following:

   a. Return the rundown request as quickly as possible, to allow processing to continue.

   b. Terminate all the task's I/O as quickly as possible. This involves closing all open files, detaching devices and killing any outstanding I/O requests.

If a handler fails to dequeue or complete a rundown request, the task will *hang* in state IR2. Other tasks that abort or exit with pending I/O will hang in state IR1.

If a handler fails to terminate one or more I/O requests, the task will eventually be placed in state IR4, since at the end of I/O rundown processing the task's I/O pending count will still be non-zero. For a task in this state, the Executive checks periodically (at every system event) to see whether the count has become zero. If so, normal exit processing is resumed. Thus, a handler might complete an I/O request after I/O rundown has terminated, enabling the task to exit normally.

Once a task is placed in state IR4, I/O rundown can proceed for other tasks in the system.

A summary of the task states involved in I/O rundown processing is given in Table 6–2.

Figure 6–1 Flow of I/O Rundown Processing

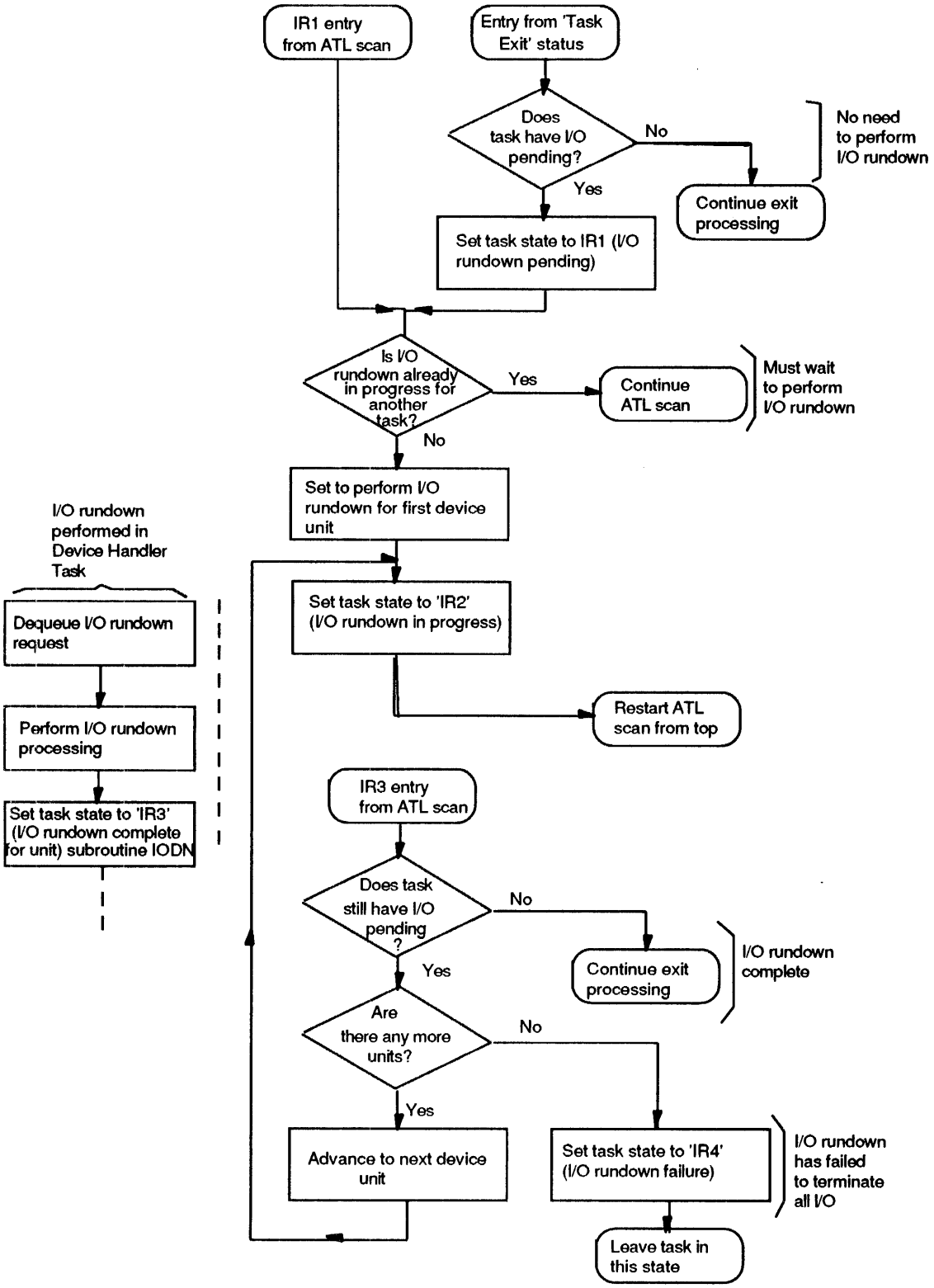


Table 6–2 I/O Rundown Task States

	State	Description
IR1	I/O rundown pending	Task is waiting for another task to complete I/O rundown processing, before it can start.
IR2	I/O rundown in progress	Task is waiting for a device handler to process a rundown request
IR3	I/O rundown done for unit	I/O rundown has been completed for the current device unit
IR4	I/O rundown failure	I/O rundown processing has been completed but the task's I/O pending count is non-zero.

See the *IAS Guide to Writing a Device Handler Task* for details about I/O rundown processing from the point of view of a device handler task.

A System Lists and Tables

```
;
; TPD -- TASK PARTITION DIRECTORY
;
; THE "TPD" IS A FIXED LIST OF ENTRIES DESCRIBING EACH PARTITION IN A
; SYSTEM.  THIS DIRECTORY IS CREATED BY THE SYSTEM CONFIGURATION
; ROUTINE (SYSGEN) CONSISTING OF ENTRIES OF THE FOLLOWING FORMAT:
;
T.PN==00 ; WD. 00 (B 00) -- PARTITION NAME (FIRST HALF)
  ; WD. 01 (B 02) -- PARTITION NAME (SECOND HALF)
T.BA==04 ; WD. 02 (B 04) -- 1/64TH BASE ADDRESS OF PARTITION (IN BYTES)
T.PZ==06 ; WD. 03 (B 06) -- 1/64TH SIZE OF PARTITION (IN BYTES)
T.FW==10 ; WD. 04 (B 10) -- PARTITION FLAGS WORD
T.TP==11 ;GLW003-(B 11) -- TIME SHARED PRIORITY
T.HP==12 ; WD. 05 (B 12) -- 1/64TH BASE ADR OF FIRST HOLE, OR ZERO IF NO HOLES.
T.RF==14 ; WD. 06 (B 12) -- ++034 LISTHEAD FOR SEND/RECEIVE POOL IN SENPAR
T.RB==16 ; WD. 07 (B 14) -- ++034 LISTHEAD BACK POINTER
T.CF==20 ; WD. 10 (B 20) -- ++034 PDR VALUE FOR SENPAR PARTITION
T.CB==22 ; WD. 11 (B 22) -- ++034 PAR VALUE FOR SENPAR PARTITION
;
T.SZ==24  ;SIZE (IN BYTES) OF TPD ENTRIES
;
; FLAGS WORD BIT DEFINITIONS:
;
TF.UC==000001 ;[00] SET IF USER CONTROLLED PARTITION,
TF.OU==000002 ;[01] SET IF OCCUPIED USER CONTROLLED PARTITION.
TF.TS==000004 ;[02] SET IF A TIME SHARED PARTITION
TF.AC==000010 ;[03] SET IF A TIME SHARED PARTITION IS ACTIVE
TF.IA==000020 ;[04] SET IF AN IAS TIMESHARING RTYPE PARTITION
TF.SG==000040 ; +++011 [05] SET IF SGN2 NEEDS TO RECONSTRUCT HOLE POINTERS

;
; MUL -- MEMORY USAGE LIST
;
; THIS LIST CONTAINS ONE ENTRY FOR EACH ALLOCATED SEGMENT OF MEMORY
; IN A T-TYPE PARTITION. IT IS PRIMARILY USED WHEN SHUFFLING MEMORY,
; SO THAT THE OCCUPANT OF EACH PART OF MEMORY CAN BE READILY IDENTIFIED.
```

```
;
  ;WD. 00 -- FORWARD LINK
  ;WD. 01-- BACKWARD LINK
M.FB==04 ;WD. 02 (B. 04) -- FLAGS BYTE
M.IO==05 ; (B. 05) -- COUNT OF ACTIVE I-O IN SEGMENT
M.NA==06 ;WD. 03 -- NODE ADDRESS (ATL OR GCD)
M.SZ==10 ;WD. 04 -- SIZE OF SEGMENT (MOD 64)
M.FS==12 ;WD. 05 -- SIZE OF HOLE ABOVE SEGMENT (MOD 64)
M.BA==14 ;WD. 06 -- BASE ADDRESS OF SEGMENT (MOD 64)
  ;WD. 07 -- SWAP LIST (IAS ONLY) ++024 NO LONGER USED
M.TS==16 ;WD. 07 -- TOTAL FREE SPACE IN THE PARTITION (LISTHEAD
  ;  MUL ENTRY ONLY).
;
; FLAG BYTE BIT DEFINITIONS:-
;
MF.GC==001 ; SEGMENT IS FOR A GLOBAL AREA (M.NA -- GCD NODE ADDR.)
  ; IF NOT SET SEGMENT IS FOR A ROOT SEGEMNT (M.NA - ATL NODE)
MF.PE==002 ; AREA CONTAINS A PARITY ERROR (THIS WILL BE SET AND THE
  ; I-O COUNT INCREMENTED TO LOCK AN AREA FROM WHICH A PARITY
  ; ERROR HAS BEEN DETECTED)
;


;
; GCD -- GLOBAL COMMON DIRECTORY
;
; THIS LIST CONTAINS THE INFORMATION REQUIRED TO CONTROL SGA'S
; CREATED BY INSTALL, AND REGIONS CREATED DYNAMICALLY BY THE CRRG$
; DIRECTIVE. IT ALSO CONTAINS ENTRIES FOR TASK PURE AREAS.
;
; THERE ARE FIVE TYPES OF ENTRY:
;
; 1. DYNAMICALLY CREATED REGIONS (G.FW = 0)
;
;  THESE ARE CREATED BY THE CRRG$ DIRECTIVE.  INITIALLY,
;  THEIR CONTENTS ARE UNDEFINED.  SUBSEQUENTLY, THEY ARE
;  MOVED TO AND FROM THE SWAP FILE.
;
; 2. INSTALLED LIBRARIES  (G.FW = GF.SG!GF.LI)
;
;  THESE ARE 'PURE', AND ARE NEVER WRITTEN OUT OF
;  MEMORY, JUST DISCARDED.  THEY ARE LOADED FROM THE
;  IMAGE FILE WHENCE THEY WERE INSTALLED.
;
; 3. PURE AREAS OF INSTALLED TASKS (G.FW = GF.SG!GF.PA)
;
;  THESE ARE EXACTLY LIKE INSTALLED LIBRARIES, EXCEPT
;  THAT THEY DO NOT HAVE A NAME.  THEY ARE CREATED BY
;  INSTALL WHEN A TASK HAS A PURE AREA.
;
; 4. INSTALLED COMMON AREAS (G.FW = GF.SG)
;
;  THESE ARE 'SWAPPED' TO AND FROM THE IMAGE FILE ON
;  THE DISK WHENCE THEY WERE INSTALLED.
;
; 5. INSTALLED REGIONS  (G.FW = G.IR)
;
;  THESE ARE INITIALLY LOADED FROM THE IMAGE FILE WHENCE
;  THEY WERE INSTALLED.  SUBSEQUENTLY, THEY ARE
;  SWAPPED USING THE SWAP FILE, SO THE ORIGINAL TASK
;  IMAGE FILE IS UNCHANGED.
;
;
  ; WD. 00 (B 00) -- FORWARD LINK
```

```
  ; WD. 01 (B 02) -- BACKWARD LINK
G.BN==04 ; WD. 02 (B 04) -- COMMON BLOCK NAME (6 CHAR IN RADIX-50, 2-WORDS)
G.BA==10 ; WD. 04 (B 10) -- 1/64TH BASE ADDRESS OF COMMON BLOCK
G.CZ==12 ; WD. 05 (B 12) -- 1/64TH SIZE OF COMMON BLOCK
G.CT==14 ; WD. 06 (B 14) -- CREATION TIME (TWO WORDS: YEAR, MONTH/DAY)
G.GS==N.SB;WD. 10 (B 20) -- GLOBAL AREA STATUS
G.SA==21 ;        (B 21) -- STARTING APR
G.OI==22 ; WD. 11 (B 22) -- OWNER IDENTIFICATION (UIC)
G.PD==24 ; WD. 12 (B 24) -- GLOBAL AREA TPD ADDRESS
  ; WD. 13 (B 26) -- RESERVED
G.DI==27 ;        (B 27) -- DISK INDICATOR
G.AC==30 ; WD. 14 (B 30) -- ACTIVE REFERENCE COUNT (BYTE)
G.IC==31 ;        (B 31) -- INSTALLED REFERENCE COUNT (BYTE)
G.SI==32 ; WD. 15 (B 32) -- SWAP FILE INDEX
G.DA==34 ; WD. 16 (B 34) -- GLOBAL AREA DISK ADDRESS
G.PR==40 ; WD. 20 (B 40) -- REGION PROTECTION MASK
G.FW==42 ; WD. 21 (B 42) -- REGION FLAGS WORD
G.TI==44 ; WD. 22 (B 44) -- REGION'S TI ASSIGNMENT
;
G.SZ==60  ;SIZE (IN BYTES) OF RDL ENTRIES
;
; FLAGS WORD BIT DEFINITIONS
;
GF.SG==000001 ;[00] SGA FLAG - SET WHEN REGION MUST
  ;   BE LOADED FROM TASK IMAGE FILE
GF.LI==000002 ;[01] LIBRARY COMMON INDICATOR -- 1:LIB  0:COM
GF.RI==000004 ;[02] LIBRARY RELOCATABILITY INDICATOR -- SET FOR PIC CODE
GF.FT==000020 ;[04] REGION HAS NOT YET BEEN LOADED - DO NOT READ FROM SWAP FILE
GF.PA==000040 ;[05] REGION IS TASK'S PURE AREA
GF.IR==000100 ;[06] REGION IS 'INSTALLED REGION'
GF.DE==000200 ;[07] REGION IS MARKED FOR DELETE
GF.TI==000400 ;[10] REGION'S NAME IS TI DEPENDENT
GF.RW==001000 ;[11] REGION IS TASK'S RW RESIDENT OVERLAY REGION
GF.PS==002000 ;[12] REGION HAS PERMANENTLY ALLOCATED SWAP SPACE  ;++014
GF.SA==004000 ;[13] GCD NODE WAS SAVED IN SYSTEM ;++029
GF.HL==010000 ;++033 [14] REGION IS TO BE LOADED HIGH
GF.MK==100000 ;++031 [17] USED BY SGN1 TO MARK GCD ENTRIES

;
; PUD -- PHYSICAL UNIT DIRECTORY
;
; THE "PUD" IS A FIXED LIST OF ENTRIES DESCRIBING EACH PHYSICAL DEVICE-
; UNIT IN A SYSTEM.  THIS LIST IS CREATED BY THE SYSTEM CONFIGURATION
; ROUTINE (SYSGEN) AND CONSISTS OF ENTRIES OF THE FOLLOWING FORMAT:
;
U.DN==00    ; WD. 00 (B 00) -- DEVICE NAME (2 ASCII CHARS)
U.UN==02    ; WD. 01 (B 02) -- UNIT NUMBER (BYTE)
U.FB==03    ;        (B 03) -- FLAGS (BYTE)
U.C1==04    ; WD. 02 (B 04) -- CHARACTERISTICS WORD ONE (DEVICE INDEPENDENT INDICA'
U.C2==06    ; WD. 03 (B 06) -- CHARACTERISTICS WORD TWO (DEVICE DEPENDENT INDICATO:
U.C3==10    ; WD. 04 (B 10) -- CHARACTERISTICS WORD THREE (DEVICE DEPENDENT INDICA'
U.C4==12    ; WD. 05 (B 12) -- CHARACTERISTICS WORD FOUR (SIZE OF BLOCK, BUFFER, L
U.AF==14    ; WD. 06 (B 14) -- ATTACH FLAG - EITHER:
    ;
    ;   1. ATL ADDRESS OF ATTACHED TASK, OR
    ;   2. PUD ADDRESS OF OWNING DEVICE (IAS
    ;      ONLY, IF UC.IAS OR UC.IEX SET)
    ;
U.RP==16    ; WD. 07 (B 16) -- REDIRECT POINTER
U.HA==20    ; WD. 10 (B 20) -- HANDLER TASK ATL NODE ADDRESS
U.XC==22    ; WD. 11 (B 22) -- COUNT OF EXPRESS REQUESTS IN QUEUE
U.RF==24    ; WD. 12 (B 24) -- UNIT REQUEST DEQUE LISTHEAD (FWD PNTR)[ OBSOLETE]
U.SL==24    ; WD. 12 (B 24) -- ADDRESS OF UIT SLOT FOR THIS DEVICE IN
```

```
    ;         HANDLER TASK
U.SC==26    ; WD. 13 (B 26) -- ADDRESS OF SCB(SHADOW CONTROL BLOCK) FOR DISK+
U.TV==30    ; WD. 14 (B 30) -- INTERRUPT TRAP VECTOR ADDRESS
U.IP==32    ; WD. 15 (B 32) -- INTERRUPT PRIORITY (IN BITS 5-7)
U.DA==34    ; WD. 16 (B 34) -- [DEVICE PAGE ADDRESS]
;
; PHYSICAL UNITS ARE CONSIDERED "VOLUMES" BY THE FILES SYSTEM, AND THE
; REMAINDER OF THE PUD ENTRY IS A "VOLUME CONTROL BLOCK".
;
; Note that the first word in the Volume Control Block has an alternate use
; for a terminal device.
;
U.VA==36    ; WD. 17 (B 36) -- ADDRESS OF VOLUME CONTROL BLOCK EXTENSION
U.LA==36    ; WD. 17 (B 36) -- Logical assignment listhead for terminals
U.UI==40    ; WD. 20 (B 40) -- USER IDENTIFICATION CODE (UIC)
U.PC==40    ;         (B 40) -- UIC PROGRAMMER CODE
U.GC==41    ;         (B 41) -- UIC GROUP CODE
U.VP==42    ; WD. 21 (B 42) -- VOLUME PROTECTION WORD
U.CH==42    ;         (B 42) -- CHARACTERISTICS FLAGS
    ;         (B 43) -- UNUSED+
U.AR==44    ; WD. 22 (B 44) -- ACCESS RIGHTS FLAGS WORD
U.DACP==46  ; WD. 23 (B 46) -- DEFAULT ACP NAME, RAD50 (FIRST WORD)
U.ACP==50   ; WD. 24 (B 50) -- EITHER:
    ;
    ;   1. STD ENTRY ADDRESS OF CURRENT ACP
    ;      (FILE STRUCTURED DEVICES)
    ;   2. CORRESPONDING UTN ADDRESS (TIMESHARING
    ;      TERMINALS)
    ;
U.TF==52    ; WD. 25 (B 52) -- TERMINAL FLAGS WORD
U.PR==52    ; WD. 25 (B 52) -- TERMINAL PRIVILEGE BYTE
U.FO==53    ;         (B 53) -- TERMINAL FORMS BYTE
U.LBH==54   ; WD. 26 (B 54) -- HIGH ORDER - TOTAL # OF BLKS FOR DEVICE
U.LBN==56   ; WD. 27 (B 56) -- LOW ORDER- TOTAL # OF BLOCKS FOR DEVICE
U.XPUD==60  ; WD. 30 (B 60) -- Virtual Address of PUD extension in device handler
;
; THIS WORD IS ONLY USED IN AN IAS SYSTEM
;
U.TS==62  ; WD. 31 (B 62) -- COUNT OF USERS OF THE VOLUME
U.MN==63  ;         (B 63) -- IAS FLAGS
;
U.SZ==64
;
; FLAGS BYTE BIT DEFINITIONS
;
UF.RH==200  ; SET WHEN HANDLER TASK IS DECLARED RESIDENT.
UF.TL==100  ; SET WHEN HANDLER TASK RECOGNIZES LOAD AND RECORD
UF.OFL==040 ; SET WHEN DEVICE IS OFFLINE
UF.CO==020  ; SET WHEN DIRECTED TO CO
UF.ACT==010 ; SET WHEN DEVICE IS ACTIVE
UF.SC==004  ; SET WHEN DISK IS THE SHADOW COPY+
UF.SD==002  ; SET WHEN DISK IS ONE OF THE SHADOW PAIR+
UF.LCK==001 ; Set to prevent all access to a device except by it's owner
  ;
;
; BIT DEFINITIONS FOR CHARACTERISTICS WORD ONE
;
UC.REC==000001 ;[00] SET IF RECORD ORIENTED DEVICE (VIZ., TT, LP, CR)
UC.CCL==000002 ;[01] SET IF CARRIAGE CONTROL DEVICE (VIZ., TT, LP)
UC.TTY==000004 ;[02] SET IF TTY DEVICE (VIZ., KSR, LA30)
UC.DIR==000010 ;[03] SET IF DEVICE IS A DIRECTORY DEVICE
UC.SDI==000020 ;[04] SET IF DEVICE IS A SINGLE DIRECTORY DEVICE
UC.SQD==000040 ;[05] SET IF DEVICE IS A SEQUENTIAL DEVICE
```

```
UC.IAS==000100 ;[06] SET IF AN INTERACTIVE IAS TERMINAL
UC.IEX==000200 ;[07] SET IF AN IAS EXCLUSIVE DEVICE
UC.INB==000400 ;+003 [08] SET IF THE DEVICE IS INTERMEDIATE BUFFERED
UC.SWL==001000 ;[09] SET IF THE DEVICE IS SOFTWARE WRITE LOCKED
UC.ISP==002000 ;[10] SET IF DEVICE IS INPUT SPOOLED
UC.OSP==004000 ;[11] SET IF DEVICE IS OUTPUT SPOOLED
UC.PSE==010000 ;[12] SET IF DEVICE IS PSEUDO DEVICE
UC.COM==020000 ;[13] SET IF DEVICE IS COMMUNICATIONS CHANNEL
UC.F11==040000 ;[14] SET IF DEVICE IS FILES-11
UC.MNT==100000 ;[15] SET IF DEVICE IS MOUNTABLE
;
; Bit definitions for characteristics word two (mass storage devices)
;
U2.WCK==000001 ;[00] SET IS READ-AFTER-WRITE CHECK REQUIRED
U2.SYD==000010 ;[03] Unit is system device
U2.MOH==000020 ;[04] SET IF DEVICE HAS MOVING HEADS
U2.RMV==000040 ;[05] SET IF DEVICE HAS REMOVABLE VOLUMES
U2.BAD==000100 ;[06] SET IF DEVICE HAS FACTORY-SUPPLIED BAD BLOCK INFO
U2.CLS==017400 ; Mask for device class code (5 bits)
U2.TYP==160000 ; Mask for device type within class code
U2.DEV==177400 ; Mask for device Class/Type bits
U2.DNS==U2.TYP ; Density bits mask for magtape devices
;
; Bits for use with the above mask
;
U2D.62==020000 ; [13] Tape drive can handle 6250 bpi
U2D.FX==040000 ; [14] Tape drive has only one density
U2D.16==100000 ; [15] Tape drive can handle 1600 bpi
;
; Supported tape density is coded as follows -
;
; Density supported   Bit(s) set
;
; 800 bpi only    U2D.FX
; 800 and 1600 bpi   U2D.16
; 1600 bpi only    U2D.16!U2D.FX
; 1600 and 6250 bpi   U2D.16!U2D.62
;
; Device Class/Type definitions
;
; For line printer the lower 2 bits are defined as follows -
;
U2.LC ==001  ; Line printer support lower case
U2.LS ==002  ; Line printer is an LS11
;
;       Device Class     Device Type
;
U2.NIL ==000  ;  00  0 Unknown Device
U2.RF1 ==001  ; 01  0 RF11 (1-8 platters)
U2.RF2 ==041  ; 01  1
U2.RF3 ==101  ; 01  2
U2.RF4 ==141  ; 01  3
U2.RF5 ==201  ; 01  4
U2.RF6 ==241  ; 01  5
U2.RF7 ==301  ; 01  6
U2.RF8 ==341  ; 01  7
;
U2.K5 ==002  ; 02  0 RK05
U2.K3 ==042  ; 02  1 RK03
U2.5F ==102  ; 02  2 RK05F
;
U2.P2 ==103  ; 03  2 RP02
U2.P3A ==203  ; 03  4 RP03A
```

```
U2.P3B ==303  ; 03  6 RP03B
;
U2.P4 ==004  ; 04  0 RP04
U2.P5 ==044  ; 04  1 RP05
U2.P6 ==104  ; 04  2 RP06
;
U2.S3 ==005  ; 05  0 RS03
U2.S4 ==045  ; 05  1 RS04
;
U2.K6 ==006  ; 06  0 RK06
U2.K7 ==046  ; 06  1 RK07
;
U2.X1 ==007  ; 07  0 RX01
;
U2.T56 ==050  ; 10  1 TU56 DECtape
;
U2.M2 ==011  ; 11  0 RM02
U2.M3 ==051  ; 11  1 RM03
U2.M5 ==111  ; 11  2 RM05
U2.M80 ==151  ; 11  3 RM80
U2.P7 ==211  ; 11  4 RP07
;
U2.L1 ==012  ; 12  0 RL01
U2.L2 ==052  ; 12  1 RL02
;
U2.T58 ==013  ; 13  0 TU58 DECtape II
;
U2.X2 ==014  ; 14  0 RX02
;
U2.A8 ==056  ; 16  1 RA80
U2.A82 ==116  ; 16  2 RA82
U2.A9 ==156  ; 16  3 RA90
U2.A6 ==216  ; 16  4 RA60
U2.A81 ==256  ; 16  5 RA81
U2.A70 ==316  ; 16  6 RA70
;
U2.F25 ==057  ; 17  1 RCF25
U2.C25 ==117  ; 17  2 RC25
;
U2.D31 ==060  ; 20  1 RD31
U2.D32 ==120  ; 20  2 RD32
U2.D33 ==220  ; 20  4 RD33
;
U2.X33 ==123  ; 23  3 RX33
U2.D54 ==163  ; 23  3 RD54
U2.D53 ==223  ; 23  4 RD53
U2.D52 ==263  ; 23  5 RD52
U2.D51 ==323  ; 23  6 RD51
U2.X50 ==363  ; 23  7 RX50
;
U2.T10 ==130  ; 30  2 TU/TE10
U2.T16 ==231  ; 31  4 TU/TE16,TU45,TU77
U2.T11 ==332  ; 32  6 TS11,TU80
;
U2.T81 ==273  ; 33  5 MU TU81
U2.T50 ==333  ; 33  6 MU TK50
U2.T70 ==373  ; 33  7 MU TK70
;
U2.LP ==036  ; 36  0 Generic line printer
U2.LS ==076  ; 36  1 Generic LS printer
```

```
UC.WCK==U2.WCK ;++008 SET IF A READ AFTER WRITE CHECK IS REQUIRED
;
; BIT DEFINITIONS FOR VOLUME CHARACTERISTICS BYTE U.CH
;
CH.OFF==200 ;VOLUME IS OFF-LINE
CH.FOR==100 ;VOLUME IS FOREIGN
CH.UNL==40 ;DISMOUNT PENDING
CH.NAT==20 ;ATTACH/DETACH NOT PERMITTED
CH.NDC==10 ;DEVICE CONTROL FUNCTIONS NOT PERMITTED
CH.LAB==1 ;VOLUME IS LABELED TAPE
;
;
; BIT DEFINITIONS FOR TERMINAL PRIVILEGE BYTE
;
UT.PR==1 ;SET IF TTY IS PRIVLEGED
UT.SL==2 ;SET IF TTY IS SLAVED
UT.LG==4 ;SET IF TERMINAL IS LOGGED ON
;
; IAS FLAG BYTE DEFINITITION
UM.PR==001 ;[01] SET IF MOUNT/DISMOUNT IN PROGRESS
UM.GB==002 ;[02] SET IF VOLUME GLOBALY MOUNTED   /IAS
UM.RT==004 ;[03] SET IF MOUNTED FOR REAL TIME
UM.TS==010 ;[04] SET IF MOUNTED FOR TIMESHARING
UM.MC==020 ;[05] SET IF MCR MOUNT
UM.RLT==200 ;[08] SET IF A TASK NEEDS TO BE RELOADED FOR THIS DEVICE
;
; Offsets in the extended PUD used by the MSCP handler
;
X.FLGS==00 ; Wd. 00 (B 00)  -- Extended PUD flags must be the first word always
X.MLUN==02 ; Wd. 01 (B 02)  -- Multiunit code
X.UNTI==04 ; Wd. 02 (B 04)  -- Unit identifier
    ; Wd. 03 (B 06)  -- X.UNTI+2 - Unit identifier (cont.)
    ; Wd. 04 (B 08.) -- X.UNTI+4 - Unit identifier (cont.)
    ; Wd. 05 (B 10.) -- X.UNTI+6 - Unit identifier (model and class)
X.SN  ==14 ; Wd. 06 (B 12.) -- Volume serial number (lo order)
    ; Wd. 07 (B 14.) -- X.SN+2 - Volume serial number (hi order)
X.TRCK==20 ; Wd. 10 (B 16.) -- Track size (LBNs per track)
X.GRP ==22 ; Wd. 11 (B 18.) -- Group size (Tracks per group)
X.CYL ==24 ; Wd. 12 (B 20.) -- Cylinder size (Groups per cylinder)
X.USVR==26 ; Wd. 13 (B 22.) -- Unit software version number
X.UHVR==27 ;     (B 23.) -- Unit hardware version number
X.RCTS==30 ; Wd. 14 (B 24.) -- Replacement Control Table size
X.RBNS==32 ; Wd. 15 (B 26.) -- Number of RBNs per track
X.RCTC==33 ;     (B 27.) -- Number of RCT copies
X.AVLH==34  ; Wd. 16 (B 28.) -- RNA listhead for available status (ST.AVL) I/O
;
X.SZ  ==40 ; Size of extended PUD entry
;
; The following bits are defined in the extended PUD flags word
;
XF.ONL==200   ; Indicates unit online in progress
XF.BBR==100   ; Host Bad Block Replacement is supported
XF.AVL==40   ; ST.AVL return status online in progress
XF.FMT==20   ; Disk is in the process of being formatted
```

```
;
; SCB -- SHADOW CONTROL BLOCK
;
; THE "SCB" IS A BLOCK OF CONTROL INFORMATION FOR USE WHEN SHADOW RECORDING
; DISKS.  THE INFORMATION HERE IS SET UP AND MAINTAINED BY THE SHADOW
; RECORDING TASKS AND BY DMQIO AND IODN.  IT IS IN THE FOLLOWING FORMAT:
;
B.PPD==00 ; WD.00 (B 00) -- PUD ADDRESS OF PRIMARY DISK
B.PSD==02 ; WD.01 (B 02) -- PUD ADDRESS OF SHADOW DISK
B.SLH==04 ; WD.02 (B 04) -- Secondary Packet Listhead
B.SLP==06 ; WD.03 (B 06) -- Primary Packet Listhead
B.LFA==10 ; WD.04 (B 10) -- CATCH-UP INFORMATION ADDRESS(LOW ORDER)
B.HFA==12 ; WD.05 (B 12) -- CATCH-UP INFO ADDRESS(HIGH ORDER)(BYTE)
B.CUS==13 ;  (B 13) -- CATCH-UP STATUS  (BYTE)
B.SST==14 ; WD.06 (B 14) -- SHADOW RECORDING STATUS
B.ATL==16 ; WD.07 (B 16) -- Shadow catchup task ATL address

;
; Flags word Bit Definition
;
BF.EBH==01 ; [00] SET IF THERE IS AN ERROR
BF.EPS==02 ; [01] SET IF THERE IS AN ERROR ON SECONDARY
  ; CLEAR IF ERROR ON PRIMARY
BF.ERW==03 ; [03] SET IF THERE IS A WRITE ERROR
  ; CLEAR IF THERE IS A READ ERROR
BF.SIP==04 ; [04] SET IF THERE IS SECONDARY I/O IN PROGRESS

;
; Catchup status byte definitions
;
BC.CIP==01 ; [00] Catchup in progress
BC.CRA==02 ; [01] Catchup task has been requested to abort


;
; LAB -- Logical Assignment Block
;
; The Logical Assignment Block is a 5 word block which contains a logical
; device name and the "physical" device it has been equated to.
;
L.LKW==00  ; WD. 0 (B 00) -- Link word
L.LDN==02  ; WD. 1 (B 02) -- Logical device name (ASCII)
L.LDU==04  ; WD. 2 (B 04) -- Logical device unit number (Binary)
    ;       (B 05) -- Unused (reserved)
L.PDN==06  ; WD. 3 (B 06) -- Physical device name corresponding to logical name
L.PDU==10  ; WD. 4 (B 10) -- Physical device unit number
    ;       (B 11) -- Unused (reserved)
```

```
;
; STD-- SYSTEM TASK DIRECTORY
;
; THE SYSTEM TASK DIRECTORY IS A MEMORY RESIDENT DIRECTORY OF ALL TASKS
; WHICH HAVE BEEN INSTALLED INTO A SYSTEM.  THIS DIRECTORY CONSISTS OF TWO
; PARTS: (1) A FIXED SIZE AREA OF ONE WORD FOR EACH TASK THAT MAY
; BE INSTALLED AT ANY TIME, AND (2) AN STD ENTRY FOR EACH TASK THAT IS
; INSTALLED.  THE FIXED SIZED AREA IS  CALLED THE "ALPHA TABLE" AND
; PROVIDES SPACE FOR AN ALPHABETICALLY ORDERED CONTIGUOUS LIST OF POINTERS
; TO STD ENTRIES TO FACILITATE SEACH FOR STD ENTRY BY TASK NAME.
; EACH STD ENTRY IS OF THE FOLLOWING FORMAT:
;
S.TN==00 ; WD. 00 (B 00) -- TASK NAME (6 CHAR IN RADIX-50, 2 WORDS)
  ; WD. 01 (B 02) -- (SECOND HALF OF TASK NAME)
S.TD==04 ; WD. 02 (B 04) -- DEFAULT TASK PARTITION (TPD ADDRESS)
S.FW==06 ; WD. 03 (B 06) -- FLAGS WORD
S.DP==10 ; WD. 04 (B 10) -- DEFAULT PRIORITY (BYTE)
S.DI==11 ;        (B 11) -- SYSTEM DISK INDICATOR (BYTE)
S.LZ==12 ; WD. 05 (B 12) -- 1/64TH SIZE OF LOAD IMAGE
S.TZ==14 ; WD. 06 (B 16) -- 1/64TH MAX TASK SIZE
S.AV==16 ; WD. 07 (B 16) -- NUMBER OF ACTIVE VERSIONS OF TASK (BYTE)
S.PV==17 ;    (B 17) -- TASK POOL LIMIT PER VERSION (BYTE)
S.PU==20 ; WD. 10 (B 20) -- TASK POOL UTILIZATION
S.RF==22 ; WD. 11 (B 22) -- RECEIVE DEQUE LISTHEAD (FWD PNTR)
S.RB==24 ; WD. 12 (B 24) -- RECEIVE DEQUE LISTHEAD (BKG PNTR)
S.DL==26 ; WD. 13 (B 26) -- LOAD IMAGE FIRST BLOCK NUMBER (32-BITS)
  ; WD. 14 (B 30)    (SECOND HALF OF DISK ADDRESS)
S.PA==32 ; WD. 15 (B 32) -- GCD NODE ADDRESS FOR PURE AREA
;
;
; THE SYSTEM DISK INDICATOR SPECIFIES WHICH I/O REQUEST QUEUE IS
; TO RECEIVE A "LOAD TASK IMAGE" REQUEST, BY PROVIDING A "PUD ENTRY INDEX".
; E.G., A ZERO WOULD INDICATE THE REQUEST QUEUE FOR THE DEVICE-UNIT
; REPRESENTED BY THE FIRST (ENTRY ZERO) PUD ENTRY.
;
; FLAGS WORD BIT DEFINITIONS:
;
SF.MK==000001 ;++031 [00] USED BY SGN1 TO MARK STD ENTRIES
SF.FX==000002 ;[01] SET WHEN TASK IS FIXED IN MEMORY
SF.RM==000004 ;[02] SET WHEN STD IS TO BE REMOVED
SF.TD==000010 ;[03] SET WHEN TASK IS DISABLED
SF.BF==000020 ;[04] SET  WHEN A TASK IS BEING FIXED IN MEMORY
SF.XT==000040 ;[05] SET WHEN A TASK IS TO BE REMOVED ON EXIT
SF.MU==000100 ;[06] SET WHEN TASK IS MULTI-USER
SF.PT==000200 ;[07] SET WHEN TASK IS A PRIVILEGED TASK
SF.NT==000400   ;[08]  NETWORK ATTRIBUTE BIT
SF.R1==001000 ;[09] RESTRICTED USAGE LEVEL ONE (BACKGROUND BATCH JOBS)
SF.XS==002000 ;[10] TASK NOT ABLE TO RECEIVE DATA OR REFERENCES
SF.XA==004000 ;[11] SET WHEN TASK IS NEVER TO BE ABORTED
SF.XD==010000 ;[12] SET WHEN TASK IS NEVER TO BE DISABLED
SF.XF==020000 ;[13] SET WHEN TASK IS NEVER TO BE FIXED IN MEMORY
SF.XC==040000 ;[14] SET WHEN TASK IS NEVER TO BE CHECKPOINTED
SF.SR==100000 ;++021 [15] SET WHEN TASK ALLOWS VSDR$ DIRECTIVE FROM ALL USERS
;
S.SIZ==32. ;SIZE OF STD IN BYTES
```

```
;
; ATL -- ACTIVE TASK LIST
;
; THE "ATL" IS A PRIORITY ORDERED DEQUE OF "ATL" NODES FOR ACTIVE TASKS
; THAT HAVE MEMORY ALLOCATED FOR THEIR EXECUTION.  THE TASKS REPRESENTED
; BY ENTRIES IN THE ATL ARE EITHER MEMORY RESIDENT, OR A REQUEST FOR THEIR
; LOADING HAS BEEN QUEUED.  THE LISTHEAD FOR THIS DEQUE IS IN THE SYSTEM
; COMMUNICATIONS AREA (SCOM), AND THE NODES ARE OF THE FOLLOWING FORMAT:
;
  ; WD. 00 (B 00) -- FORWARD LINKAGE
  ; WD. 01 (B 02) -- BACKWARD LINKAGE
  ; WD. 02 (B 04) -- NODE ACCOUNTING WORD (STD ENTRY ADR OF REQUESTOR)
A.RQ==N.AW
A.TI==N.TI;WD. 03 (B 06) -- TI IDENTIFICATION - PUD ADDRESS
A.RP==10 ; WD. 04 (B 10) -- TASK'S RUN PRIORITY (BYTE)
A.IR==11 ;        (B 11) -- TASK I/O IN PROCESS COUNT (BYTE)
A.IN==12 ; WD. 05 (B 12) -- TASK I/O PENDING COUNT (BYTE)
A.CS==13 ;        (B 13) -- SAVED STATUS OF CHECKPOINTED TASK
A.MT==14 ; WD. 06 (B 14) -- TASK MARK TIME PENDING COUNT (BYTE)
A.CP==15 ;        (B 15) -- SAVED PRIORITY OF CHECKPOINTED TASK (BYTE)
A.HA==16 ; WD. 07 (B 16) -- 1/64TH REAL ADR OF LOAD IMAGE
A.TS==N.SB;WD. 10 (B 20) -- TASK STATUS (BYTE)
A.AS==21 ;        (B 21) -- AST INDICATOR (PREVIOUS STATUS) BYTE
A.TD==22 ; WD. 11 (B 22) -- SYSTEM TASK DIRECTORY (STD) ENTRY ADDRESS
A.EF==24 ; WD. 12 (B 24) -- TASK'S EVENT FLAGS (1-32)
  ; WD. 13 (B 26) -- (SECOND HALF OF TASK'S EVENT FLAGS)
A.FM==30 ; WD. 14 (B 30) -- TASK'S EVENT FLAGS MASKS (64-BITS)
  ; WD. 15 (B 32) -- (SECOND WORD OF FLAGS MASK)
  ; WD. 16 (B 34) -- (THIRD WORD OF FLAGS MASK)
  ; WD. 17 (B 36) -- (FOURTH WORD OF FLAGS MASK)
  ;
  ; THE EVENT FLAG MASKS AT A.FM ARE USED FOR VARIOUS PURPOSES
  ; BY THE EXEC, TO RECORD INFORMATION ABOUT THE STATE OF A TASK.
  ; THE SIGNIFICANCE OF THESE WORDS DEPENDS ON THE STATE OF THE
  ; TASK:
  ;
  ; 1. UP TO FIRST TIME LOAD (STATES LRP, LRQ, LRS)
  ;
  ; A.FM+0 PUD ADDRESS OF DEVICE TO LOAD TASK FROM
  ; A.FM+2 ADDRESS OF STL NODE FOR THIS TASK, OR ZERO
  ; A.FM+4 UIC FOR TASK TO RUN, OR 0 IF NOT SPECIFIED
  ; A.FM+6 ATL ADDRESS OF TASK WHICH REQUESTED THIS ONE,
  ;  IF REQUESTED BY EXEC$ OR FIX$
  ;
  ; 2. WAITING OR STOPPED FOR SINGLE GROUP OF EVENT FLAGS
  ;     (STATES WF0, WF1, WF2, WF3, ST0, ST1, ST2, ST3)
  ;
  ; A.FM+0 MASK FOR FLAGS BEING WAITED FOR IN RELEVANT
  ;  EVENT FLAG WORD (A.EF+0, A.EF+2, .COMEF, .COMEF+2)
  ;
  ; 3. WAITING OR STOPPED FOR ALL GROUPS OF EVENT FLAGS (STATES WF4, ST4)
  ;
  ; A.FM+0 MASK FOR FLAGS 1-16  (A.EF+0)
  ; A.FM+2 MASK FOR FLAGS 17-32 (A.EF+2)
  ; A.FM+4 MASK FOR FLAGS 33-48 (.COMEF)
  ; A.FM+6 MASK FOR FLAGS 49-64 (.COMEF+2)
  ;
  ; 4. WAITING FOR EXECUTIVE SEMAPHORE (STATE WSM)
  ;
  ; A.FM+0 MASK FOR SEMAPHORE BEING WAITED FOR
  ;
  ; 5. AFTER TASK EXIT (STATES EXT, STN)
  ;
```

```
  ; A.FM+0 REASON FOR EXIT (LO BYTE), EXIT FLAGS (HI BYTE):
  ;
  ;  BIT 8 (000400)  SET IF TKTN REQUIRED
  ;  BIT 9 (001000) SET IF I/O RUNDOWN REQUIRED
  ;  BIT 10 (002000) SET IF TASK EXITED WITH VALID STATUS
  ;
  ; A.FM+2 TASK EXIT STATUS
  ;
  ; 6. WAITING FOR DIRECTIVE (STATE WDI)
  ;
  ;    MEANING DEPENDS ON PARTICULAR DIRECTIVE.  CURRENTLY THIS IS
  ;    USED FOR:
  ;
  ; EXEC$, FIX$:
  ;
  ;    A.FM+6 PRESET TO -03, ERROR CODE FOR 'INSUFFICIENT
  ;   MEMORY'
  ;
  ;
  ; 7. DIRECTIVE FAILED (STATE DIF)
  ;
  ;    MUST BE SET BY CODE WHICH PUTS TASK IN THIS STATE TO:
  ;
  ; A.FM+6 ERROR CODE TO RETURN TO TASK'S DSW
  ;
  ;  NOTE THAT A.FM+0 CANNOT BE USED BECAUSE THIS STATE
  ;  OCCURS FOR A TASK AFTER IT HAS EXITED, WHEN
  ;   .TKTN. IS REQUESTED
  ;
A.PD==40 ; WD. 20 (B 40) -- TASK'S RUN PARTITION (TPD ADDRESS)
A.AF==42 ; WD. 21 (B 42) -- AST DEQUE LISTHEAD (FWD POINTER)
A.AB==44 ; WD. 22 (B 44) -- AST DEQUE LISTHEAD (BKWD POINTER)
A.SA==46 ; WD. 23 (B 46) -- SWAP ADDRESS
A.TZ==50 ; WD. 24 (B 50) -- CURRENT TASK SIZE  ++023
A.TF==52 ; WD. 25 (B 52 -- TASK FLAGS
A.SD==54 ; WD.26 (B. 54) -- ALLOCATION FACTOR
  ;     DISC ADDRESS IN A.IA
A.QI==55 ;        (B. 55) -- COUNT OF ACTIVE AND QUEUED I-O
A.SW==56 ; WD.27 (B. 56) -- COUNT OF SWAP I-O
A.SS==57 ; (B. 57) -- SAVE STATUS FOR IAS SUSPEND
;
; TASK STATUS VALUES ARE DESCRIBED AT 'ASXDT'
;
;
AF.CP==001 ; SET WHEN TASK IS CHECKPOINTED
AF.SA==002 ; Set when task was running in super mode prior to AST
AF.AD==004 ; SET WHEN TASK AST RECOGNITION IS INHIBITED
AF.CD==010 ; SET WHEN CHECKPOINTING IS DISABLED
AF.MC==020 ; SET WHEN TASK IS MARKED FOR CHECKPOINTING
AF.KA==040 ; SET WHEN TASK HAS A KERNAL AST QUEUED+
AF.IO==100 ; SET WHEN TASK HAS AN I/O COMPLETION EVENT IN ITS AST QUEUE
AF.PF==200 ; SET WHEN THERE IS A POTENTIAL POWER FAIL AST  ; +++010
AF.RR==400 ; SET WHEN POTENTIAL RECEIVE BY REFERENCE AST
AF.BF==1000 ; SET WHEN A TASK IS TO BE FIXED
AF.FX==2000 ; SET WHEN A TASK IS FIXED
AF.AS==4000 ; SET WHEN AN AST HAS BEEN DECLARED
AF.RA==10000 ; SET WHEN THERE IS A POTENTIAL RECEIVE AST
AF.RL==20000 ; SET IF TASK NEEDS TO BE RELOADED
AF.IA==40000 ; SET IF THE TASK IS IAS CONTROLLED
AF.TR==100000 ; SET IF THE TASK IS DOING TT READ
;
A.SIZ==48. ;SIZE OF ATL IN BYTES
;
```

```
;
; IF A TIMESHARING TASK THE ATL WILL BE 8 WORDS LARGER AN CONTAIN
; THE FOLLOWING ADDITIONAL INFORMATION
;
A.TUF==60; WD. 30 (B 60) -- UTL FORWARD POINTER
A.TUB==62; WD. 31 (B 62) -- UTL BACKWARD POINTER
A.TFW==64; WD. 32 (B 64) -- TIMESHARING FLAGS WORD
A.TST==66; WD. 33 (B 66) -- TIMESHARING STATUS BYTE
A.TSV==67;    (B 67) -- STATUS SAVE
A.JN==70 ; WD. 34 (B 70) -- JOB NODE ADDRESS  ++023
A.TAI==72; WD. 35 (B 72) -- ACCOUNTING STATE   ++032
  ; VALUE: 0 - TASK LOADING     (NO INFO)   ++032
  ;  2 - TASK SWAPPED OUT (INFO IN UJN) ++032
  ;  4 - TASK IN MEMORY   (INFO IN HEADER) ++032
  ;  6 - TASK EXITING     (INFO IN ATL E.TAC)++032
  ;    (B 73) -- (SPARE)    ++032
A.TQU==74; WD. 36 (B 74) -- QUANTUM
A.TLV==76; WD. 37 (B 76) -- UTL LEVEL LIST HEAD
;
A.TSIZ==64.
;
; TIMESHARING TASK FLAGS WORD BIT DEFINITIONS
;
AT.NL == 001 ;FIRST TIME LOAD
AT.TR == 002 ;SET IF TASK IS RESIDENT
AT.TL == 004 ;SET IF TASK IS TO BE LOADED
AT.IA == 010 ;SET IF INSTALL IS ACTIVE (OR TO BE RUN)
;  ;++018 UNUSED
;  ;++018 UNUSED
AT.IB == 100 ; SET IF TASK TO BE ABORTED WHEN INSTALL IS COMPLETE
AT.LS == 200 ; SET IF LUNS NEED TO BE REASSIGNED
AT.DS == 400 ; DELETE STD NODE ON EXIT
AT.TH == 1000 ; TEMPORARY HIGH-PRIORITY, USED TO FORCE LOADING ;++015
AT.BT == 2000 ; BATCH TASK (CLI OR USER TASK)
AT.SA == 4000 ; TASK NON-SWAPPABLE FOR ABORT
AT.TA == 10000 ; TASK IS ON THE ATL OR IS BEING INSTALLED
AT.LD == 20000 ; TASK IS LOADING
AT.DB == 40000 ; TASK IS TO BE INTERRUPTED FOR DEBUGGING AID
AT.HP == 100000 ; TASK IS TO BE RUN AT HIGH ATL PRIORITY     ;++015
;
;
;
;
; TIMESHARING TASK STATUS BYTE VALUES
;
; THESE VALUES ARE USED IN A.TST TO CONTROL TASK OPERATION WITH RESPECT
; TO THE TIMESHARING SCHEDULER (TSSHED).
;
JS.RUN ==00  ; TASK RUNNABLE
;
JS.RSD ==02  ; TASK TO BE SUSPENDED
;
JS.SUS ==04  ; TASK IS SUSPENDED
;
JS.ABT ==06  ; TASK TO BE ABORTED
;
JS.NEW ==10  ; TASK NEW TO SCHEDULER
;
JS.EXT ==12  ; TASK EXITED (BUT NOT YET PROCESSED BY TCP)
;
JS.LOD ==14  ; TASK TO BE LOADED
;
JS.CON ==16  ; TASK TO BE CONTINUED
```

```
;
JS.NW2 ==20   ; TASK NEW AFTER INSTALL
;
JS.EXX ==22   ; TASK EXITING, TCP QIO PENDING
;
JS.FIN ==24   ;++026 TASK EXITED AND PROCESSED BY TCP (IE UJN RELEASED)

;
; TIMESHARING ATL LINKAGE
;
; FOR TIMESHARING TASKS THE ATL IS ALSO LINKED INTO LEVELS ACCORDING
; TO THE PREVIOUS ACTIVITY OF THE TASK.  MOST SERVICING OF TIMESHARING
; TASK'S ATLS IS DONE WITH A REGISTER ADDRESSING THE UTL POINTER.
; THE FOLLOWING OFFSETS ARE DEFINED SO THAT THE WHOLE ATL CAN BE
; REFERENCED WHEN A REGISTER POINTS TO THE UTL (A.TUF)
;
;
X.RQ==A.RQ-A.TUF
X.TI==A.TI-A.TUF
X.RP==A.RP-A.TUF
X.IR==A.IR-A.TUF
X.IN==A.IN-A.TUF
X.CS==A.CS-A.TUF
X.MT==A.MT-A.TUF
X.CP==A.CP-A.TUF
X.HA==A.HA-A.TUF
X.NA==A.HA-A.TUF
X.TS==A.TS-A.TUF
X.AS==A.AS-A.TUF
X.TD==A.TD-A.TUF
X.EF==A.EF-A.TUF
X.FM==A.FM-A.TUF
X.PD==A.PD-A.TUF
X.AF==A.AF-A.TUF
X.AB==A.AB-A.TUF
X.SA==A.SA-A.TUF
X.TZ==A.TZ-A.TUF      ; ++023
X.TF==A.TF-A.TUF
X.SD==A.SD-A.TUF
X.QI==A.QI-A.TUF
X.SW==A.SW-A.TUF
X.SS==A.SS-A.TUF
X.UF==A.TUF-A.TUF
X.UB==A.TUB-A.TUF
X.FW==A.TFW-A.TUF
X.ST==A.TST-A.TUF
X.SV==A.TSV-A.TUF
X.JN==A.JN-A.TUF      ; ++023
X.AI==A.TAI-A.TUF      ; ++032
X.QU==A.TQU-A.TUF
X.LV==A.TLV-A.TUF
```

```
;
; TASK HEADER OFFSETS
;
; THESE ARE DEFINED IN A MACRO IN THE MACRO LIBRARY, WHICH IS
; IN THE FILE [311,2]TSKIMG.MAC.  THIS FILE SHOULD BE CONSULTED
; FOR REFERENCE.
;
 .MCALL HDRSY$
 HDRSY$ DEF$G
;
; DEFINE SYMBOLS REQUIRED BY RSX ACCOUNTING
;
H.DEV==2    ;NUMBER OF DEVICES CURRENTLY ACCESSED
H.TMA==4    ;TIME ALLOWED TASK IN TICKS (CPU)
H.TM1==6    ;
H.IDA==8.    ;UNIQUE I.D. NUMBER
H.NAM==10.    ;NAME OF TASK
H.NM1==12.
H.AUC==14.    ;ACCOUNTING UIC
H.CPU==16.    ;CPU TIME RECORD
H.CP1==18.
H.CP2==20.
H.CP3==22.
H.CP4==24.
H.CP5==26.
H.COR==28.    ;CORE USE RECORD
H.CO1==30.
H.ETM==32.    ;START TIME RECORD
H.DV1==48.    ;DEVICE RECORDS START


;
; REGION AND WINDOW DESCRIPTOR BLOCK DEFINITIONS
;
 .MCALL RDBDF$,WDBDF$
 RDBDF$ DEF$G   ; DEFINE RDB OFFSETS
 WDBDF$ DEF$G   ; AND WDB OFFSETS

;
; E.XXX OFFSETS
;
E.JB ==0   ;++011 JOB ID
E.SIZ ==E.JB+2 ;++011 TASK SIZE
E.TIM ==E.SIZ+2 ;++011 CPU TIME (2 WORDS)
;CONTINUED BELOW.............
;
;
;  DEFINITION OF E.XXX OFFSETS    ;++009
; INTO THE ATL AFTER EXIT OF TASK. TSS1 USES ;++009
; THE ATL FOR PASSING EXIT INFO TO TCP   ;++009
; AFTER EXIT, PARTICULARLY EXIT WITH STATUS. ++009
;
; THESE OFFSETS ARE ALSO USED BY PI.SEV DRB'S ++011
;
E.TR ==E.TIM+4   ;++010/09 REASON FOR EXIT
E.TS ==E.TR+2   ;++009 TASK'S EXIT STATUS
E.TPS ==E.TS+2   ;++009 TASKS PS
E.TPC ==E.TPS+2;   ;++009 TASK'S PC
E.TR0 ==E.TPC+2   ;++009 AND REGISTERS
E.TR1 ==E.TR0+2   ;++009
E.TR2 ==E.TR1+2   ;++009
E.TR3 ==E.TR2+2   ;++009
E.TR4 ==E.TR3+2   ;++009
```

```
E.TR5 ==E.TR4+2  ;++009
E.TSP ==E.TR5+2  ;++009
E.TAC ==E.TSP+2  ; ++027 CPU TIME (2 WORDS) AT TASK EXIT TIME

 .MCALL EXST$  ; DEFINE THE EXIT STATUS CODES, GLOBALLY
 $$$GLB=0
 EXST$
;
.IF LT E.TR-<3*2> ;++010 MUSTN'T BE EARLIER THAN WORD 3
 .ERROR   ;INVALID ATL OFFSETS
.ENDC
;
.IF GT E.TAC+2-<24.*2> ; ++027 ++010 MUSTN'T OVERSTEP THE END OF THE NODE
 .ERROR   ;INVALID ATL OFFSETS
.ENDC


;
; SFL -- SWAP FILE LIST
;
; THE "SFL" IS THE LIST OF SWAP FILES CURRENTLY AVAILABLE TO THE SYSTEM.
; IT IS USED BY THE SWAP FILE ALLOCATION/DEALLOCATION ROUTINES, IN
; CONJUNCTION WITH THE SWAP FILE BITMAP.  IT IS ALSO USED WHEN
; TRANSLATING A SWAP FILE BLOCK NUMBER INTO A PUD ADDRESS (DEVICE)
; AND DISK LBN.  THE ENTRIES ARE IN ASCENDING ORDER OF
; SWAP FILE.

  ; WD. 00 (B 00) -- FORWARD LINKAGE
  ; WD. 01 (B 02) -- BACKWARD LINKAGE
S.FID == 4 ; WD. 02 (B 04) -- FILE ID OF THIS SWAP FILE (THREE WORDS)
  ; WD. 03 (B 06)
  ; WD. 04 (B 10)
S.LBN ==12 ; WD. 05 (B 12) -- START LBN OF SWAP FILE
  ; WD. 06 (B 14)
S.PUD ==16 ; WD. 07 (B 16) -- PUD ADDRESS OF SWAP DEVICE
S.LEN ==20 ; WD. 10 (B 20) -- TOTAL NO OF BITS IN BITMAP FOR FILE
S.ALC ==22 ; WD. 11 (B 22) -- NUMBER OF BLOCKS ALLOCATED IN THIS FILE
S.RND ==24 ; WD. 12 (B 24) -- NUMBER OF BLOCKS IN BITMAP BUT NOT
  ;      ACTUALLY IN FILE (I.E. DIFFERENCE
  ;      BETWEEN S.LEN AND ACTUAL FILE LENGTH)
S.WFB ==25 ;  (B 25) -- FLAGS BYTE

;
;
; FLAG BYTE DEFINTIONS
;

SW.BAD ==001 ; FILE CONTAINS BAD BLOCKS
SW.DE ==002 ; FILE IS MARKED FOR DELETE
SW.DV ==004 ; FILE IS ON DEDICATED SWAP VOLUME
SW.RT ==010 ; FILE IS RESERVED FOR REALTIME USAGE
```

```
;
; UTL -- USER TASK LIST
;
; THIS LIST IS A DEQUE OF ENTRIES USED BY THE SCHEDULER
; TO FIND WHICH TASK TO RUN.
; IT IS DIVIDED INTO A NUMBER OF LEVELS WHICH
; DETERMINE THE PRIORITY OF THE TASKS.
; EACH ENTRY IN THE DEQUE CONTAINS THE LIST HEAD
; OF A DEQUE OF JOB NODES WHICH BELONG TO THAT
; LEVEL.
; THE SCHEDULER CAN PROMOTE AND DEMOTE TASKS BETWEEN
; LEVELS ON THE BASIS OF THEIR ACTIVITY HISTORY,
; BY UNLINKING NODES FROM ONE LEVEL AND RELINKING
; THEM INTO ANOTHER.
; JOBS IN THE LEVEL 1 UTL ENTRY GET HIGHEST
; PRIORITY SERVICE FROM THE SCHEDULER.
; THE MAXIMUM NUMBER OF LEVELS IS SPECIFIED AT SYSGEN
; TIME.
;
; UTL ENTRY OFFSETS :-
Z.NL == 00 ;WD. 00 -- ADDRESS OF NEXT LEVEL
Z.PL == 02 ;WD. 01 -- ADDRESS OF PREVIOUS LEVEL
Z.FJ == 04 ;WD. 02 -- ADDRESS OF FIRST JOB NODE FOR LEVEL
Z.LJ == 06 ;WD. 03 -- ADDR. OF LAST JOB NODE FOR LEVEL
  ;WD. 04 -- DUMMY FLAG WORD (LOOKS LIKE A UJN)
  ;WD. 05 -- DUMMY STATUS WORD
Z.NE == 14 ;WD. 06 -- (B 14) -- NO. OF ENTRIES FOR LEVEL
Z.FG == 15 ;           (B 15) -- FLAGS BYTE
Z.NT == 16 ;WD. 07 -- ROBIN POINTER FOR LEVEL
Z.TF == 20 ;WD. 10 -- TIME FACTOR FOR THIS LEVEL
Z.LD == 22 ;WD. 11 -- NEXT JOB TO LOAD
Z.SP == 24 ;WD. 12 (B 24) -- SPARE
Z.LV == 25 ; (B 25) -- LEVEL NUMBER
Z.NS == 26 ;WD. 13 -- NEXT TASK TO SWAP
;
;
;
; FLAGS BYTE (Z.FG) DEFINITIONS:-
;
ZB.SD == 001 ; TASK SCHEDULED AT THIS LEVEL
ZB.BT == 002 ; BATCH SCHEDULING LEVEL (MUST BE THE BOTTOM LEVEL IF SET)
;
Z.SIZ == 40 ; SIZE OF UTL IN BYTES
;
```

```
;
; IRQ -- I/O REQUEST QUEUE
;
; THE "IRQ" IS A PRIORITY ORDERED DEQUE OF I/O REQUEST NODES WITH ITS
; LISTHEAD IN THE PUD ENTRY OF THE PHYSICAL UNIT FOR WHICH THE I/O
; REQUEST WAS QUEUED.  EACH PHYSICAL UNIT HAS ITS OWN I/O REQUEST QUEUE.
; I/O REQUEST NODES ARE CREATED AND QUEUED PRIMARILY BY THE "QUEUE I/O"
; DIRECTIVE.  HOWEVER, THE EXEC ALSO CREATES I/O REQUESTS TO:
; (1) LOAD A TASK IMAGE, (2) RECORD A TASK IMAGE [CHECKPOINTING], AND
; (3) TO RUNDOWN I/O ON AN EXIT'ED TASK.  I/O REQUEST NODES ARE OF
; THE FOLLOWING FORMAT.
;
  ; WD. 00 (B 00) -- FORWARD LINKAGE
  ; WD. 01 (B 02) -- BACKWARD LINKAGE
  ; WD. 02 (B 04) -- NODE ACCOUNTING WORD (STD ENTRY ADR OF REQUESTOR)
R.TD==N.AW
R.AT==06 ; WD. 03 (B 06) -- ATL NODE OF REQUESTOR ***
R.PR==10 ; WD. 04 (B 10) -- PRIORITY (BYTE)
R.DP==11 ;        (B 11) -- DPB SIZE (BYTE) ***
R.LU==12 ; WD. 05 (B 12) -- LOGICAL UNIT NUMBER (BYTE)
R.FN==13 ;        (B 13) -- EVENT FLAG NUMBER (BYTE)
R.FC==14 ; WD. 06 (B 14) -- I/O FUNCTION CODE
R.SB==16 ; WD. 07 (B 16) -- VIRTUAL ADDRESS OF STATUS BLOCK
R.AE==20 ; WD. 10 (B 20) -- VIRTUAL ADDRESS OF AST SERVICE ENTRY
R.UI==22 ; WD. 11 (B 22) -- USER IDENTIFICATION CODE
R.PC==22 ;        (B 22) -- PROGRAMMER CODE
R.GC==23 ;        (B 23) -- GROUP CODE
R.PB==24 ; WD. 12 (B 24) -- PARAMETER #1
  ; WD. 13 (B 26) -- PARAMETER #2
  ; WD. 14 (B 30) -- PARAMETER #3
  ; WD. 15 (B 32) -- PARAMETER #4
  ; WD. 16 (B 34) -- PARAMETER #5
  ; WD. 17 (B 36) -- PARAMETER #6
R.PD==40 ; WD. 20 (B 40) -- PUD POINTER FOR THIS REQUEST
R.EL==42 ; WD. 21 (B 42) -- ERROR LOG BUFFER POINTER/FLAG
R.WA==44 ; WD. 22 (B 44) -- FLAG BYTE FOR EXEC
R.HF==45 ; WD. 22 (B 45) -- WORK AREA FOR DEVICE HANDLERS (Handler Flags)
  ; WD. 23 (B 46) -- WORK AREA FOR DEVICE HANDLERS
  ; WD. 24 (B 50) -- WORK AREA FOR DEVICE HANDLERS
R.IA==52 ; WD. 25 (B 52) -- ASR3 VALUE FOR BUFFER BASE(=-1 FOR SCOMM)
R.IB==54 ; WD. 26 (B 54) -- EITHER:
  ;
  ;    1. INTERMEDIATE BUFFER ADDRESS (RSX
  ;       INTERMEDIATE BUFFERED DEVICES)
  ;    2. TPD ADDRESS FOR PARTITION (IAS
  ;       EXEC LOAD/RECORD REQUESTS)
  ;    3. ADDRESS OF BLOCK LOCK NODE (FILE
  ;       STRUCTURED DEVICES)
  ;
R.UB==56 ; WD. 27 (B 56) -- EITHER:
  ;
  ;    1. USER BUFFER ADDRESS (RSX INTERMEDIATE
  ;       BUFFERED DEVICES)
  ;    2. MUL NODE ADDRESS (IAS)
;
; FOR EXECUTIVE I/O REQUESTS A LARGER NODE IS USED TO ALOOW
; TRANSFERS GREATER THAN 32K-32 WORDS
;
R.BA==60 ; WD. 30 (B 60) -- BASE ADDRESS OF COMPLETE TRANSFER (MOD 64)
R.TB==62 ; WD. 31 (B 62) -- BASE ADDRESS CURRENT TRANSFER (MOD 64)
R.TS==64 ; WD. 32 (B 64) -- TRANSFER SIZE (MOD 64)
R.BN==66 ; WD. 33 (B 66) -- CURRENT BLOCK NUMBER (HI)
  ; WD. 34 (B 70) -- CURRENT BLOCK NUMBER (LO)
```

```
;
RS.BLK==127.  ; MAXIMUM NUMBER OF 256. WORD DISC BLOCKS IF
   ; NOT LAST TRANSFER
;
RS.32W==RS.BLK*10 ; MAXIMUM TRANSFER SIZE IN 32. WORD BLOCKS IF
   ; NOT LAST TRANSFER
RS.MAX==RS.32W+7 ; MAXIMUM TRANSFER SIZE IN 32. WORD BLOCKS IF
   ; LAST TRANSFER
;
; THE LOW ORDER THREE-BITS OF THE I/O FUNCTION CODE ARE USED BY THE SYSTEM
; AS FOLLOWS:
;
RF.IT==000001 ;[0] -- RESERVED FOR FUTURE USE
RF.XR==000002 ;[1] -- "EXPRESS REQUEST"
RF.IR==000004 ;[2] -- RESERVED FOR FUTURE USE
RF.GC==000040 ;[5] -- GCD RECORD REQU. NODE INDICATOR
; IAS EXECUTIVE I-O FLAGS
;
RW.LK==200 ; SET IF MEMORY LOCKED FOR REQUEST (MUL ADDRESS IN R.UB)
RW.ML==100 ; SET IF NODE (GCD OR ATL) ADDRESS STORED IN R.UB
RW.IA==010 ; SET IF AN IAS SWAP REQUEST
RW.SW==020 ; SET IF THE SWAP COUNT IS INCREMENTED FOR REQUEST
RW.SP==004 ; SET IF REQUEST IS TO OUTPUT SPOOLED DEVICE ;++017/16
;
R.SIZ == 60 ; SIZE OF TASK REQUEST NODE IN BYTES
;
R.XSIZ== 100 ; SIZE OF EXECUTIVE REQUEST NODE IN BYTES
;
; Flags used with the internal handlers' work area (R.HF)
;
RHF.AB== 1    ; Handler's per request aborted bit
RHF.RN== 2    ; Release request node address for error log
RHF.MS== 4    ; Multicopy Structure function in progress
RHF.EL== 10    ; BBR request to log error (ER$LOG)          050
RHF.BB== 200    ; Request owned by HIBBR task

;
; *** WHENEVER AN I/O REQUEST IS QUEUED BY THE "QUEUE I/O" DIRECTIVE, THE
; DPB SIZE AND THE REQUESTOR'S ATL NODE ADDRESS ARE RECORDED IN THE I/O
; REQUEST NODE.  WHENEVER AN I/O REQUEST IS QUEUED AS A RESULT OF ANOTHER
; DIRECTIVE (VIZ., "REQUEST" CAUSING A TASK IMAGE TO BE LOADED), THE DPB
; SIZE AND THE REQUESTOR'S ATL NODE ADDRESS ARE SET TO ZERO.  THUS, BOTH
; BOTH THE DPB SIZE AND THE ATL NODE ADDRESS ARE ALSO "EXEC REQUEST"
; INDICATORS.
```

```
;
; UMR -- ALLOCATION AND DEALLOCATION
;
; THE FOLLOWING FIVE WORD BLOCK MUST BE PRESENT IN EACH HANDLER THAT
; CALLS THE UMR ALLOCATION ROUTINE.  THE "MAP REGISTER BLOCK" IS
; DEFINED AS FOLLOWS:
;
M.RN==0 ;++028 WD. 00 -- REQUEST NODEA ADDR OF OWNER
M.PW==2 ;++020 WD. 01 -- PRE ALLOCATED SLOT/LENGTH WORD
M.DF==4 ;++020 WD. 02 -- NUMBER OF PRE-ALLOCATED UMRS (CAN BE 0)
M.SL==6 ;++020 WD. 03 -- SLOT/LENGTH WORD (SET BY ..ALMR)
M.UL==10;++020 WD. 04 -- LOW 16 BITS OF UNIBUS ADDRESS (SET BY ..ALMR)
M.UH==12;++020 WD. 05 -- HIGH 2 BITS OF UNIBUS ADDRESS (SET BY ..ALMR)
;
;
; SYMBOLS FOR UMR SUPPORT
;
ON.UM==40  ;GLW002-UMR REQUESTED SYMBOL
ON.70==100  ;GLW002 SYSGENED FOR A 70
ON.44==200  ; ++030 ++028 SYSGENNED FOR A 44
ON.22==20  ;GLW002-22-BIT ON
ON.CSM==10  ;ENABLE CSM INSTRUCTION
ON.QB==4  ;22 BIT Q-BUS (Q22)
;
; THE FOLLOWING BITS ARE DEFINED FOR USE BY THE EXECUTIVE
; WHEN ENABLING/DISABLING DATA SPACE FOR THE VARIOUS
; MODES.
;

ON.KD==4  ; ENABLE KERNAL D-SPACE
ON.SD==2  ; ENABLE SUPERVISOR D-SPACE
ON.UD==1  ; ENABLE USER D-SPACE

.UMASK==74  ;GLW002-SET IN UMR TO CLEAR
.UMRAD==170200  ;GLW002-ADDRESS OF UMRS


;
; CKQ -- CLOCK QUEUE
;
; THE CLOCK QUEUE IS A DEQUE CONSISTING OF ONE NODE FOR EACH OPERATION
; SCHEDULED TO BE PERFORMED AT SOME FUTURE TIME.  A "SCHEDULE DELTA-
; TIME" IN THE FIRST NODE (IF ANY) OF THE CLOCK QUEUE IS DECREMENTED
; AT EACH CLOCK TICK UNTIL THE NODE "COMES DUE", AT WHICH TIME THE
; INDICATED OPERATION IS PERFORMED.  CLOCK QUEUE NODES ARE LINKED
; IN THE ORDER IN WHICH THEY WILL COME DUE, AND THE SCHEDULE DELTA-TIME
; IN EACH NODE (EXCEPT THE FIRST) IS RELATIVE TO THE SCHEDULE TIME
; OF THE PREVIOUS CLOCK QUEUE NODE.  CLOCK QUEUE NODES ARE OF THE
; FOLLOWING FORMAT.
;
  ; WD. 00 -- FORWARD LINKAGE
  ; WD. 01 -- BACKWARD LINKAGE
  ; WD. 02 -- NODE ACCOUNTING WORD (STD ENTRY ADR OF REQUESTOR)
C.TD==N.AW
C.AT==06 ; WD. 03 -- ATL NODE ADDRESS OF REQUESTOR
C.SD==10 ; WD. 04 -- SCHEDULE DELTA IN TICKS (64-BITS)
  ; WD. 05 -- (LOWER ORDER HALF OF SCHEDULE DELTA)
C.RT==14 ; WD. 06 -- REQUEST TYPE INDICATOR
;
C.FM==16 ; WD. 07 -- [MT] FLAG MASK (BIS SRC)
C.FA==20 ; WD. 10 -- [MT] FLAGS WORD ADR (BIS DST ADR)
C.FN==22 ; WD. 11 -- [MT] EVENT FLAG NUMBER
C.AE==24 ; WD. 12 -- [MT] VIRTUAL ADDRESS OF AST SERVICE ENTRY
  ; (5 UNUSED WORDS)
;
```

```
C.RI==16 ; WD. 07 -- [TS] RESCHEDULE INTERVAL IN TICKS (64-BITS)
  ; WD. 10 -- [TS] (LOW ORDER HALF OF RESCHEDULE INTERVAL)
C.R2==22 ; WD. 11 -- [TS] STD ENTRY ADR OF REQUESTED TASK (R2 FOR '.REQS')
C.R3==24 ; WD. 12 -- [TS] TPD ENTRY ADR, OR ZERO (R3 FOR '.REQS')
C.R4==26 ; WD. 13 -- [TS] RUN PRIORITY, OR ZERO (R4 FOR '.REQS')
C.UI==30 ; WD. 14 -- [TS] UIC INDICATOR FOR '.REQS'
C.TI==32 ; WD. 15 -- [TS] TI IDENTIFICATION FOR '.REQS'
  ; (2 UNUSED WORDS)
;
; [MT] -- MARK TIME NODE ENTRIES
; [TS] -- TASK SCHEDULING NODE ENTRIES
;
; REQUEST TYPE INDICATORS:
CF.TS==000400
;
; C.RT LOW BYTE ZERO DENOTES MARK-TIME ENTRY
;  HIGH BYTE = 0 DENOTES TASK REQUEST
;  CF.TS SET DENOTES TIMESHARING SCHEDULER ENTRY
;
; C.RT LOW BYTE NON-ZERO DENOTES TASK SCHEDULING ENTRY
;  = 1  SINGLE-SHOT REQUEST
;  = 2  PERIODIC RESCHEDULING REQUEST
;
; 0 -- MARK TIME
.IF DF TSCH
CF.SS==000001   ; INDICATES SINGLE SHOT SCHEDULE
CF.RS==000002   ; INDICATES PERIODIC RESCHEDULING
CF.SL==000400   ; INDICATES A TIME SLICE ENTRY
.ENDC
;
; NOTE -- THE CLOCK QUEUE SCAN ROUTINE IN "CANCEL SCHEDULED REQUESTS"
; ASSUMES TASK SCHEDULING IF NON-ZERO REQUEST TYPE INDICATOR.

;
; ASQ -- AYNCHRONOUS SYSTEM TRAP QUEUE
;
; THE "ASQ" IS A DEQUE (FIFO), WITH LISTHEAD IN ATL ENTRIES, CONSISTING
; OF ONE NODE FOR EACH AST (ASYNCHRONOUS SYSTEM TRAP) TO BE EXECUTED FOR
; THE TASK DEFINED BY THE STD ENTRY.  ASQ NODES ARE OF THE FOLLOWING
; FORMAT.
;
  ; WD. 00 -- FORWARD LINKAGE
  ; WD. 01 -- BACKWARD LINKAGE
  ; WD. 02 -- ACCOUNTING WORD (STD ENTRY ADDRESS OF CHARGED TASK)
Y.TT==06 ; WD. 03 -- AST TYPE & NUMBER OF PARAMETERS **
Y.AE==10 ; WD. 04 -- AST ENTRY POINT
Y.P1==12 ; WD. 05 -- AST PARAMETER 1
  ; WD. 05 -- AST PARAMETER 2
  ; WD. 06 -- AST PARAMETER 3
  ; .... ETC.
;
; ** THE AST TYPE & NUMBER OF PARAMETER DEFINITIONS ARE AS FOLLOWS:
;
YF.MT==0+<400*1> ;MARK-TIME (PARAMETER: EVENT FLAG NUMBER)
YF.IC==1+<400*1> ;I/O COMPLETION (PARAMETER: STATUS BLOCK ADDRESS)
YF.FE==2+<400*2> ;F.P. EXCEPTION (PARAMETERS: EXCEPTION CODE & ADDRESS)
YF.PR==3+<400*0> ;POWER RECOVERY (NO PARAMETERS)
YF.RE==4+<400*0> ;RECEIVE QUEUE'D (NO PARAMETERS)
YF.RR==5+<400*0> ;RECEIVE BY REFERENCE QUEUED (NO PARAMETERS)
YF.SC==6+<400*1> ;SPAWN TASK COMPLETION (PARAMETER: STATUS BLOCK ADR)
YF.PC==7+<400*3> ;COMMUNICATIONS AST
YF.TT==10+<400*0> ;TERMINAL AST
YF.KA==11+<400*4> ;KERNAL AST (PARAMETER: 1ST 2 BLANK,3RD MAPPING VALUE,
```

```
   ;  4TH PROCEDURE ADDRESS.)+

;
; SRQ -- SEND/RECEIVE QUEUE
; RRQ -- SEND/RECEIVE BY REFERENCE QUEUE
;
; THE "SRQ" IS A DEQUE (FIFO), WITH LISTHEAD IN STD ENTRIES, CONSISTING
; ONE NODE FOR EACH BLOCK OF DATA "SENT" (VIA "SEND" OR "SEND & REQUEST"
; DIRECTIVES) TO THE TASK DEFINED BY THE STD ENTRY.  RQS NODES ARE OF
; THE FORMAT DEFINED BELOW.
;
; THE 'RRQ' IS A SINGLE LIST WHICH CONTAINS ALL DATA PACKETS FOR
; ALL SEND/RECEIVE BY REFERENCE INFORMATION AT ANY TIME.  THE FORMAT
; OF THE NODES IS VERY SIMILAR TO SEND DATA NODES, AND THE SAME OFFSETS
; ARE USED WHERE POSSIBLE.
;
    ; WD. 00 -- FORWARD LINKAGE
    ; WD. 01 -- BACKWARD LINKAGE
D.SI==N.AW ; WD. 02 (B 04) -- SENDER ID (NAW)
D.TI==N.TI ; WD. 03 (B 06) -- TI INDICATOR
D.PR==10   ; WD. 04 (B 10) -- PRIORITY OF SEND
D.BS==11   ;        (B 11) -- BUFFER SIZE (WORDS)
D.D1==12   ; WD. 05 (B 12) -- FIRST WORD OF DATA BLOCK
;
; RRQ-SPECIFIC OFFSETS
;
D.TD==10   ; WD. 04 (B 10) -- STD ADDRESS OF RECEIVER TASK
D.RD==12   ; WD. 05 (B 12) -- RDL ADDRESS OF REGION BEING SENT
D.FM==14   ; WD. 06 (B 14) -- FLAG MASK BIT TO SET IN EVENT FLAGS
D.FA==16   ; WD. 07 (B 16) -- ADDRESS TO SET FLAG MASK (IN SENDER'S ATL NODE)
D.WO==20   ; WD. 10 (B 20) -- WINDOW OFFSET FOR RECEIVER
D.WL==22   ; WD. 11 (B 22) -- WINDOW LENGTH FOR RECEIVER
D.FB==24   ; WD. 12 (B 24) -- FLAGS BYTE (CONTAINS ACCESS PERMISSION BITS)
    ;        (B 25) -- RESERVED
D.DB==26   ; WD. 13 (B 26) -- EIGHT WORD DATA BUFFER
;
D.SIZ==60  ; SIZE OF NODE IN BYTES

;
; STL -- SPAWN TASK LIST
;
; THIS LIST CONTAINS THE ONE NODE FOR EACH SPAWNED TASK (I.E. TASK INITIATED
; BY THE SPWN$ DIRECTIVE).  IN ADDITION, IF A COMMAND LINE WAS ISSUED
; WITH THE DIRECTIVE, THE NODE CONTAINS THE COMMAND LINE UNTIL IT
; IS PICKED BY THE GMCR$ DIRECTIVE.
;
; A SPAWNED TASK HAS A POINTER IN ITS HEADER TO ITS STL NODE, SO THAT THERE
; IS NO NEED TO SEARCH THE STL TO FIND THE RELEVANT NODE.  THE ONLY
; PURPOSE OF THE STL IS TO ALLOW THE EXEC TO FIND ALL TASKS SPAWNED BY
; ANOTHER TASK WHEN IT EXITS, SO THAT THE LINKAGE CAN BE UNDONE.
;
;
;  ; WD. 00 -- FORWARD LINKAGE
;  ; WD. 01 -- BACKWARD LINKAGE
  ; WD. 02 -- NODE ACCOUNTING WORD
M.RATL==6 ; WD. 03 -- ATL ADDRESS OF REQUESTING TASK
M.SATL==10 ; WD. 04 -- ATL ADDRESS OF SPAWNED TASK
M.EAST==12 ; WD. 05 -- ADDRESS OF AST ROUTINE TO INITIATE WHEN
  ;     SPAWNED TASK EXITS
M.ESB==14 ; WD. 06 -- EXIT STATUS BLOCK ADDRESS
M.EFN==16 ; WD. 07 -- EVENT FLAG TO SET WHEN SPAWNED TASK EXITS
M.SFB==17 ; WD. 08 -- FLAG BYTE
;
```

```
; THE COMMAND LINE PART, WHOSE DEFINITION FOLLOWS, EXISTS ONLY
; WHILE A COMMAND LINE IS ACTUALLY PRESENT.  IT IS DEALLOCATED WHEN
; THE SPAWNED TASK PERFORMS A GMCR$ DIRECTIVE.  IT IS ALLOCATED
; B E F O R E   THE ACTUAL STL NODE, SO OFFSETS INTO IT ARE
; NEGATIVE.  THIS IS DONE TO REDUCE THE NODE POOL FRAGMENTATION
; WHICH WOULD OTHERWISE OCCUR IF A SINGLE NODE IS DEALLOCATED WHEN
; A TASK EXITS, AFTER IT MAY HAVE CREATED OTHER, MORE
; PERMANENT NODES.
;
M.MCRL==-120 ;         -- START OF COMMAND LINE
M.SBC==-1 ;         -- LENGTH IN BYTES OF COMMAND LINE
;
M.SIZ==140 ; SIZE OF NODE
M.SIZ1==120 ; SIZE OF COMMAND LINE PORTION OF NODE
;
; FLAG BITS
;
MF.CML==001 ; [01] COMMAND LINE PRESENT
MF.RMC==002 ; [02] RE-REQUEST REQUESTING TASK ON EXIT

;
; MCR -- MCR COMMAND BUFFER
;
; THIS DATA STRUCTURE EXISTS ONLY FOR COMPATIBILITY WITH EARLIER VERSIONS
; OF IAS AND RSX11D.  IT MAY BE USED TO PASS A COMMAND LINE TO A TASK, ALTHOUGH
; THE CORRECT WAY TO DO THIS IS VIA THE SPWN$ DIRECTIVE.  THE MCR
; COMMAND BUFFER MAY BE REMOVED AT ANY TIME FROM THE SYSTEM AND SHOULD
; NOT BE USED IN ANY NEW CODE.  FURTHERMORE, EXISTING CODE WHICH USES
; IT SHOULD BE MODIFIED TO USE SPWN$ AT THE FIRST OPPORTUNITY.
;
  ; WD. 00 -- FORWARD LINKAGE
  ; WD. 01 -- BACKWARD LINKAGE
  ; WD. 02 -- NODE ACCOUNTING WORD
M.TN==6  ; WD. 03 -- SECOND HALF OF MCR TASK NAME
M.TI==10 ; WD. 04 -- TI ADDRESS OF MCR FUNCTION
M.BC==12 ; WD. 05 -- NO OF BYTES IN COMMAND LINE
M.BF==14 ; WD. 06 -- START OF DATA AREA IN BUFFER
```

B QIOMAC

```
1           .TITLE  QIOMAC - QIOSYM MACRO DEFINITION
2       ;
3       ; DATE OF LAST MODIFICATION:
4       ;
5       ;       J.A. KASSON     5-FEB-80
6       ;
7       ;
8       ; ***** ALWAYS UPDATE THE FOLLOWING TWO LINES TOGETHER
9               .IDENT  /0340/
10              QI.VER=0340
11
12      ;
13      ; COPYRIGHT (C) 1980
14      ; DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
15      ;
16      ; THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY ON A
17      ; SINGLE COMPUTER SYSTEM AND MAY  BE  COPIED   ONLY  WITH  THE
18      ; INCLUSION OF THE ABOVE COPYRIGHT NOTICE.  THIS SOFTWARE,  OR
19      ; ANY OTHER COPIES THEREOF, MAY NOT BE PROVIDED  OR  OTHERWISE
20      ; MADE AVAILABLE TO ANY OTHER PERSON   EXCEPT FOR  USE ON SUCH
21      ; SYSTEM AND TO ONE WHO AGREES TO THESE LICENSE  TERMS.  TITLE
22      ; TO AND OWNERSHIP OF THE SOFTWARE SHALL AT ALL  TIMES  REMAIN
23      ; IN DEC.
24      ;
25      ; THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT
26      ; NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL
27      ; EQUIPMENT CORPORATION.
28      ;
29      ; DEC ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF
30      ; ITS SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DEC.
31      ;
32      ;
33      ; PETER H. LIPMAN 1-OCT-73
34      ;
35      ;+
36      ; MACRO TO DEFINE STANDARD QUEUE I/O DIRECTIVE FUNCTION VALUES
37      ; AND IOSB RETURN VALUES.  TO INVOKE AT ASSEMBLY TIME (WITH LOCAL
38      ; DEFINITION) USE:
39      ;
40      ;       QIOSY$          ;DEFINE SYMBOLS
41      ;
42      ; TO OBTAIN GLOBAL DEFINITION OF THESE SYMBOLS USE:
43      ;
44      ;       QIOSY$ DEF$G    ;SYMBOLS DEFINED GLOBALLY
45      ;
46      ; THE MACRO CAN BE CALLED ONCE ONLY AND THEN
47      ; REDEFINES ITSELF AS NULL.
48      ;-
49
50              .MACRO  QIOSY$  $$$GBL,$$$MSG
51              .IIF    IDN,<$$$GBL>,<DEF$G>,   .GLOBL  QI.VER
52              .IF     IDN,<$$$MSG>,<DEF$S>
53              $$$MAX=0
54              $$MSG=1
55              .IFF
```

```
56            $$MSG=0
57            .ENDC
58            .MCALL  IOERR$
59            IOERR$  $$$GBL          ;I/O ERROR CODES FROM HANDLERS, FCP, FCS
60            .MCALL  DRERR$
61            DRERR$  $$$GBL          ;DIRECTIVE STATUS WORD ERROR CODES
62            .IF     DIF,<$$$MSG>,<DEF$S>
63            .MCALL  FILIO$
64            FILIO$  $$$GBL          ;DEFINE GENERAL I/O FUNCTION CODES
65            .MCALL  SPCIO$
66            SPCIO$  $$$GBL          ;DEVICE DEPENDENT I/O FUNCTION CODES
67            .MACRO  QIOSY$  ARG,ARG1,ARG2   ;RECLAIM MACRO STORAGE
68            .ENDM   QIOSY$
69            .ENDC
70            .ENDM   QIOSY$
71
72
73      ;
74      ; DEFINE THE ERROR CODES RETURNED BY DEVICE HANDLER AND FILE PRIMITIVES
75      ; IN THE FIRST WORD OF THE I/O STATUS BLOCK
76      ; THESE CODES ARE ALSO RETURNED BY FILE CONTROL SERVICES (FCS) IN THE
77      ; BYTE F.ERR IN THE FILE DESCRIPTOR BLOCK (FDB)
78      ;   THE BYTE F.ERR+1 IS 0 IF F.ERR CONTAINS A HANDLER OR FCP ERROR CODE.
79      ;
80            .MACRO  IOERR$  $$$GBL
81            .MCALL  .IOER.,DEFIN$
82            .IF     IDN,<$$$GBL>,<DEF$G>
83            ...GBL=1
84            .IFF
85            ...GBL=0
86            .ENDC
87            .IIF    NDF,$$MSG,$$MSG=0
88
89
90      ;
91      ; SYSTEM STANDARD CODES, USED BY EXECUTIVE AND DRIVERS
92      ;
93
94            .IOER.  IE.BAD,-01.,<BAD PARAMETERS>
95            .IOER.  IE.IFC,-02.,<INVALID FUNCTION CODE>
96            .IOER.  IE.DNR,-03.,<DEVICE NOT READY>
97            .IOER.  IE.VER,-04.,<PARITY ERROR ON DEVICE>
98            .IOER.  IE.ONP,-05.,<HARDWARE OPTION NOT PRESENT>
99            .IOER.  IE.SPC,-06.,<ILLEGAL USER BUFFER>
100           .IOER.  IE.DNA,-07.,<DEVICE NOT ATTACHED>
101           .IOER.  IE.DAA,-08.,<DEVICE ALREADY ATTACHED>
102           .IOER.  IE.DUN,-09.,<DEVICE NOT ATTACHABLE>
103           .IOER.  IE.EOF,-10.,<END OF FILE DETECTED>
104           .IOER.  IE.EOV,-11.,<END OF VOLUME DETECTED>
105           .IOER.  IE.WLK,-12.,<WRITE ATTEMPTED TO LOCKED UNIT>
106           .IOER.  IE.DAO,-13.,<DATA OVERRUN>
107           .IOER.  IE.SRE,-14.,<SEND/RECEIVE FAILURE>
108           .IOER.  IE.ABO,-15.,<REQUEST TERMINATED>
109           .IOER.  IE.PRI,-16.,<PRIVILEGE VIOLATION>
110           .IOER.  IE.RSU,-17.,<SHARABLE RESOURCE IN USE>
111           .IOER.  IE.OVR,-18.,<ILLEGAL OVERLAY REQUEST>
112           .IOER.  IE.BYT,-19.,<ODD BYTE COUNT (OR VIRTUAL ADDRESS)>
113           .IOER.  IE.BLK,-20.,<LOGICAL BLOCK NUMBER TOO LARGE>
114           .IOER.  IE.MOD,-21.,<INVALID UDC MODULE #>
115           .IOER.  IE.CON,-22.,<UDC CONNECT ERROR>
116           .IOER.  IE.BBE,-56.,<BAD BLOCK ON DEVICE>
117           .IOER.  IE.STK,-58.,<NOT ENOUGH STACK SPACE (FCS OR FCP)>
118           .IOER.  IE.FHE,-59.,<FATAL HARDWARE ERROR ON DEVICE>
```

```
119             .IOER.  IE.EOT,-62.,<END OF TAPE DETECTED>
120             .IOER.  IE.OFL,-65.,<DEVICE OFF LINE>
121             .IOER.  IE.BCC,-66.,<BLOCK CHECK, CRC, OR FRAMING ERROR>
122
123
124     ;
125     ; FILE PRIMITIVE CODES
126     ;
127
128             .IOER.  IE.NOD,-23.,<CALLER'S NODES EXHAUSTED>
129             .IOER.  IE.DFU,-24.,<DEVICE FULL>
130             .IOER.  IE.IFU,-25.,<INDEX FILE FULL>
131             .IOER.  IE.NSF,-26.,<NO SUCH FILE>
132             .IOER.  IE.LCK,-27.,<LOCKED FROM READ/WRITE ACCESS>
133             .IOER.  IE.HFU,-28.,<FILE HEADER FULL>
134             .IOER.  IE.WAC,-29.,<ACCESSED FOR WRITE>
135             .IOER.  IE.CKS,-30.,<FILE HEADER CHECKSUM FAILURE>
136             .IOER.  IE.WAT,-31.,<ATTRIBUTE CONTROL LIST FORMAT ERROR>
137             .IOER.  IE.RER,-32.,<FILE PROCESSOR DEVICE READ ERROR>
138             .IOER.  IE.WER,-33.,<FILE PROCESSOR DEVICE WRITE ERROR>
139             .IOER.  IE.ALN,-34.,<FILE ALREADY ACCESSED ON LUN>
140             .IOER.  IE.SNC,-35.,<FILE ID, FILE NUMBER CHECK>
141             .IOER.  IE.SQC,-36.,<FILE ID, SEQUENCE NUMBER CHECK>
142             .IOER.  IE.NLN,-37.,<NO FILE ACCESSED ON LUN>
143             .IOER.  IE.CLO,-38.,<FILE WAS NOT PROPERLY CLOSED>
144             .IOER.  IE.DUP,-57.,<ENTER - DUPLICATE ENTRY IN DIRECTORY>
145             .IOER.  IE.BVR,-63.,<BAD VERSION NUMBER>
146             .IOER.  IE.BHD,-64.,<BAD FILE HEADER>
147             .IOER.  IE.EXP,-75.,<FILE EXPIRATION DATE NOT REACHED>
148             .IOER.  IE.BTF,-76.,<BAD TAPE FORMAT>
149             .IOER.  IE.ALC,-84.,<ALLOCATION FAILURE>
150             .IOER.  IE.ULK,-85.,<UNLOCK ERROR>
151             .IOER.  IE.WCK,-86.,<WRITE CHECK FAILURE>
152             .IOER.  IE.DSQ,-90.,<DISK QUOTA EXCEEDED>
153
154     ;
155     ; FILE CONTROL SERVICES CODES
156     ;
157
158             .IOER.  IE.NBF,-39.,<OPEN - NO BUFFER SPACE AVAILABLE FOR FILE>
159             .IOER.  IE.RBG,-40.,<ILLEGAL RECORD SIZE>
160             .IOER.  IE.NBK,-41.,<FILE EXCEEDS SPACE ALLOCATED, NO BLOCKS>
161             .IOER.  IE.ILL,-42.,<ILLEGAL OPERATION ON FILE DESCRIPTOR BLOCK>
162             .IOER.  IE.BTP,-43.,<BAD RECORD TYPE>
163             .IOER.  IE.RAC,-44.,<ILLEGAL RECORD ACCESS BITS SET>
164             .IOER.  IE.RAT,-45.,<ILLEGAL RECORD ATTRIBUTES BITS SET>
165             .IOER.  IE.RCN,-46.,<ILLEGAL RECORD NUMBER - TOO LARGE>
166             .IOER.  IE.2DV,-48.,<RENAME - 2 DIFFERENT DEVICES>
167             .IOER.  IE.FEX,-49.,<RENAME - NEW FILE NAME ALREADY IN USE>
168             .IOER.  IE.BDR,-50.,<BAD DIRECTORY FILE>
169             .IOER.  IE.RNM,-51.,<CAN'T RENAME OLD FILE SYSTEM>
170             .IOER.  IE.BDI,-52.,<BAD DIRECTORY SYNTAX>
171             .IOER.  IE.FOP,-53.,<FILE ALREADY OPEN>
172             .IOER.  IE.BNM,-54.,<BAD FILE NAME>
173             .IOER.  IE.BDV,-55.,<BAD DEVICE NAME>
174             .IOER.  IE.NFI,-60.,<FILE ID WAS NOT SPECIFIED>
175             .IOER.  IE.ISQ,-61.,<ILLEGAL SEQUENTIAL OPERATION>
176             .IOER.  IE.NNC,-77.,<NOT ANSI 'D' FORMAT BYTE COUNT>
177
178     ;
179     ; NETWORK ACP CODES
180     ;
181
```

```
182             .IOER.  IE.AST,-80.,<NO AST SPECIFIED IN CONNECT>
183             .IOER.  IE.NNN,-68.,<NO SUCH NODE>
184             .IOER.  IE.NFW,-69.,<PATH LOST TO PARTNER>;THIS CODE MUST BE ODD
185             .IOER.  IE.BLB,-70.,<BAD LOGICAL BUFFER>
186             .IOER.  IE.TMM,-71.,<TOO MANY OUTSTANDING MESSAGES>
187             .IOER.  IE.NDR,-72.,<NO DYNAMIC SPACE AVAILABLE>
188             .IOER.  IE.CNR,-73.,<CONNECTION REJECTED>
189             .IOER.  IE.TMO,-74.,<TIMEOUT ON REQUEST>
190             .IOER.  IE.NNL,-78.,<NOT A NETWORK LUN>
191
192     ;
193     ; ICS/ICR ERROR CODES
194     ;
195        .IOER.  IE.NLK,-79.,<TASK NOT LINKED TO SPECIFIED ICS/ICR INTERRUPTS>
196        .IOER.  IE.NST,-80.,<SPECIFIED TASK NOT INSTALLED>
197        .IOER.  IE.FLN,-81.,<DEVICE OFFLINE WHEN OFFLINE REQUEST WAS ISSUED>
198
199
200     ;
201     ; TTY ERROR CODES
202     ;
203
204             .IOER.  IE.IES,-82.,<INVALID ESCAPE SEQUENCE>
205             .IOER.  IE.PES,-83.,<PARTIAL ESCAPE SEQUENCE>
206
207
208     ;
209     ; RECONFIGURATION CODES
210     ;
211
212             .IOER.  IE.ICE,-47.,<INTERNAL CONSISTANCY ERROR>
213             .IOER.  IE.ONL,-67.,<DEVICE ONLINE>
214
215     ;
216     ; PCL ERROR CODES
217     ;
218
219             .IOER.  IE.NTR,-87.,<TASK NOT TRIGGERED>
220             .IOER.  IE.REJ,-88.,<TRANSFER REJECTED BY RECEIVING CPU>
221             .IOER.  IE.FLG,-89.,<EVENT FLAG ALREADY SPECIFIED>
222
223
224     ;
225     ; SUCCESSFUL RETURN CODES---
226     ;
227
228             DEFIN$  IS.PND,+00.     ;OPERATION PENDING
229             DEFIN$  IS.SUC,+01.     ;OPERATION COMPLETE, SUCCESS
230             DEFIN$  IS.RDD,+02.     ;FLOPPY DISK SUCCESSFUL COMPLETION
231                                     ;OF A READ PHYSICAL, AND DELETED
232                                     ;DATA MARK WAS SEEN IN SECTOR HEADER
233             DEFIN$  IS.TNC,+02.     ;(PCL) SUCCESSFUL TRANSFER BUT MESSAGE
234                                     ;TRUNCATED (RECEIVE BUFFER TOO SMALL).
235             DEFIN$  IS.BV,+05.      ;(A/D READ) AT LEAST ONE BAD VALUE
236                                     ;WAS READ (REMAINDER MAY BE GOOD).
237                                     ;BAD CHANNEL IS INDICATED BY A
238                                     ;NEGATIVE VALUE IN THE BUFFER.
239
240
241     ;
242     ; TTY SUCCESS CODES
243     ;
244
```

```
245             DEFIN$  IS.CR,<15*400+1>  ;CARRIAGE RETURN WAS TERMINATOR
246             DEFIN$  IS.ESC,<33*400+1> ;ESCAPE (ALTMODE) WAS TERMINATOR
247             DEFIN$  IS.CC,<3*400+1> ;CONTROL-C WAS TERMINATOR
248             DEFIN$  IS.ESQ,<233*400+1>  ;ESCAPE SEQUENCE WAS TERMINATOR
249             DEFIN$  IS.PES,<200*400+1>  ;PARTIAL ESCAPE SEQUENCE TERMINATOR
250             DEFIN$  IS.EOT,<4*400+1>  ;EOT WAS TERMINATOR (BLOCK MODE INPUT)
251             DEFIN$  IS.TAB,<11*400+1> ;TAB WAS TERMINATOR (FORMS MODE INPUT)
252             DEFIN$  IS.TMO,+2.      ;REQUEST TIMED OUT
253
254
255     ; ******
256     ;
257     ; THE NEXT AVAILABLE ERROR NUMBER IS: -90.
258     ; ALL LOWER NUMBERS ARE IN USE !!
259     ;
260     ; *****
261             .IF     EQ,$$MSG
262             .MACRO  IOERR$  A
263             .ENDM   IOERR$
264             .ENDC
265             .ENDM   IOERR$
266
267     ;
268     ; DEFINE THE DIRECTIVE ERROR CODES RETURNED IN THE DIRECTIVE STATUS WORD
269     ;
270     ; FILE CONTROL SERVICES (FCS) RETURNS THESE CODES IN THE BYTE F.ERR
271     ; OF THE FILE DESCRIPTOR BLOCK (FDB).  TO DISTINGUISH THEM FROM THE
272     ; OVERLAPPING CODES FROM HANDLER AND FILE PRIMITIVES, THE BYTE
273     ; F.ERR+1 IN THE FDB WILL BE NEGATIVE FOR A DIRECTIVE ERROR CODE.
274     ;
275             .MACRO  DRERR$  $$$GBL
276             .MCALL  .QIOE.,DEFIN$
277             .IF     IDN,<$$$GBL>,<DEF$G>
278             ...GBL=1
279             .IFF
280             ...GBL=0
281             .ENDC
282             .IIF    NDF,$$MSG,$$MSG=0
283     ;
284     ; STANDARD ERROR CODES RETURNED BY DIRECTIVES IN THE DIRECTIVE STATUS WORD
285     ;
286             .QIOE.  IE.UPN,-01.,<INSUFFICIENT DYNAMIC STORAGE>
287             .QIOE.  IE.INS,-02.,<SPECIFIED TASK NOT INSTALLED>
288             .QIOE.  IE.PTS,-03.,<PARTITION TOO SMALL FOR TASK>
289             .QIOE.  IE.UNS,-04.,<INSUFFICIENT DYNAMIC STORAGE FOR SEND>
290             .QIOE.  IE.ULN,-05.,<UN-ASSIGNED LUN>
291             .QIOE.  IE.HWR,-06.,<DEVICE HANDLER NOT RESIDENT>
292             .QIOE.  IE.ACT,-07.,<TASK NOT ACTIVE>
293             .QIOE.  IE.ITS,-08.,<DIRECTIVE INCONSISTENT WITH TASK STATE>
294             .QIOE.  IE.FIX,-09.,<TASK ALREADY FIXED/UNFIXED>
295             .QIOE.  IE.CKP,-10.,<ISSUING TASK NOT CHECKPOINTABLE>
296             .QIOE.  IE.TCH,-11.,<TASK IS CHECKPOINTABLE>
297             .QIOE.  IE.RBS,-15.,<RECEIVE BUFFER IS TOO SMALL>
298             .QIOE.  IE.PRI,-16.,<PRIVILEGE VIOLATION>
299             .QIOE.  IE.RSU,-17.,<RESOURCE IN USE>
300             .QIOE.  IE.NSW,-18.,<NO SWAP SPACE AVAILABLE>
301             .QIOE.  IE.ILV,-19.,<ILLEGAL VECTOR SPECIFIED>
302     ;
303     ;
304     ;
305             .QIOE.  IE.AST,-80.,<DIRECTIVE ISSUED/NOT ISSUED FROM AST>
306             .QIOE.  IE.MAP,-81.,<ILLEGAL MAPPING SPECIFIED>
307             .QIOE.  IE.IOP,-83.,<WINDOW HAS I/O IN PROGRESS>
```

```
308             .QIOE.  IE.ALG,-84.,<ALIGNMENT ERROR>
309             .QIOE.  IE.WOV,-85.,<ADDRESS WINDOW ALLOCATION OVERFLOW>
310             .QIOE.  IE.NVR,-86.,<INVALID REGION ID>
311             .QIOE.  IE.NVW,-87.,<INVALID ADDRESS WINDOW ID>
312             .QIOE.  IE.ITP,-88.,<INVALID TI PARAMETER>
313             .QIOE.  IE.IBS,-89.,<INVALID SEND BUFFER SIZE ( .GT. 255.)>
314             .QIOE.  IE.LNL,-90.,<LUN LOCKED IN USE>
315             .QIOE.  IE.IUI,-91.,<INVALID UIC>
316             .QIOE.  IE.IDU,-92.,<INVALID DEVICE OR UNIT>
317             .QIOE.  IE.ITI,-93.,<INVALID TIME PARAMETERS>
318             .QIOE.  IE.PNS,-94.,<PARTITION/REGION NOT IN SYSTEM>
319             .QIOE.  IE.IPR,-95.,<INVALID PRIORITY ( .GT. 250.)>
320             .QIOE.  IE.ILU,-96.,<INVALID LUN>
321             .QIOE.  IE.IEF,-97.,<INVALID EVENT FLAG ( .GT. 64.)>
322             .QIOE.  IE.ADP,-98.,<PART OF DPB OUT OF USER'S SPACE>
323             .QIOE.  IE.SDP,-99.,<DIC OR DPB SIZE INVALID>
324     ;
325     ; SUCCESS CODES FROM DIRECTIVES - PLACED IN THE DIRECTIVE STATUS WORD
326     ;
327             DEFIN$  IS.CLR,0        ;EVENT FLAG WAS CLEAR
328                                     ;FROM CLEAR EVENT FLAG DIRECTIVE
329             DEFIN$  IS.SET,2        ;EVENT FLAG WAS SET
330                                     ;FROM SET EVENT FLAG DIRECTIVE
331             DEFIN$  IS.SPD,2        ;TASK WAS SUSPENDED
332     ;
333     ;
334             .IF     EQ,$$MSG
335             .MACRO  DRERR$  A
336             .ENDM   DRERR$
337             .ENDC
338             .ENDM   DRERR$
339
340     ;
341     ; DEFINE THE GENERAL I/O FUNCTION CODES - DEVICE INDEPENDENT
342     ;
343             .MACRO  FILIO$  $$$GBL
344             .MCALL  .WORD.,DEFIN$
345             .IF     IDN,<$$$GBL>,<DEF$G>
346             ...GBL=1
347             .IFF
348             ...GBL=0
349             .ENDC
350     ;
351     ; GENERAL I/O QUALIFIER BYTE DEFINITIONS
352     ;
353             .WORD.  IQ.X,001,000    ;NO ERROR RECOVERY
354             .WORD.  IQ.Q,002,000    ;QUEUE REQUEST IN EXPRESS QUEUE
355             .WORD.  IQ.S,004,000    ;SYNONYM FOR IQ.UMD
356             .WORD.  IQ.UMD,004,000  ;USER MODE DIAGNOSTIC STATUS REQUIRED
357     ;
358     ; EXPRESS QUEUE COMMANDS
359     ;
360
361             .WORD.  IO.KIL,012,000  ;KILL CURRENT REQUEST
362             .WORD.  IO.RDN,022,000  ;I/O RUNDOWN
363             .WORD.  IO.UNL,042,000  ;UNLOAD I/O HANDLER TASK
364             .WORD.  IO.LTK,050,000  ;LOAD A TASK IMAGE FILE
365             .WORD.  IO.RTK,060,000  ;RECORD A TASK IMAGE FILE
366             .WORD.  IO.SET,030,000  ;SET CHARACTERISTICS FUNCTION
367     ;
368     ; GENERAL DEVICE HANDLER CODES
369     ;
370             .WORD.  IO.WLB,000,001  ;WRITE LOGICAL BLOCK
```

```
371             .WORD.  IO.RLB,000,002  ;READ LOGICAL BLOCK
372             .WORD.  IO.LOV,010,002  ;LOAD OVERLAY (DISK DRIVER)
373             .WORD.  IO.ATT,000,003  ;ATTACH A DEVICE TO A TASK
374             .WORD.  IO.DET,000,004  ;DETACH A DEVICE FROM A TASK
375     ;
376     ; DIRECTORY PRIMITIVE CODES
377     ;
378             .WORD.  IO.FNA,000,011  ;FIND FILE NAME IN DIRECTORY
379             .WORD.  IO.RNA,000,013  ;REMOVE FILE NAME FROM DIRECTORY
380             .WORD.  IO.ENA,000,014  ;ENTER FILE NAME IN DIRECTORY
381     ;
382     ; FILE PRIMITIVE CODES
383     ;
384             .WORD.  IO.CLN,000,007  ;CLOSE OUT LUN
385             .WORD.  IO.ULK,000,012  ;UNLOCK BLOCK
386             .WORD.  IO.ACR,000,015  ;ACCESS FOR READ
387             .WORD.  IO.ACW,000,016  ;ACCESS FOR WRITE
388             .WORD.  IO.ACE,000,017  ;ACCESS FOR EXTEND
389             .WORD.  IO.DAC,000,020  ;DE-ACCESS FILE
390             .WORD.  IO.RVB,000,021  ;READ VIRITUAL BLOCK
391             .WORD.  IO.WVB,000,022  ;WRITE VIRITUAL BLOCK
392             .WORD.  IO.EXT,000,023  ;EXTEND FILE
393             .WORD.  IO.CRE,000,024  ;CREATE FILE
394             .WORD.  IO.DEL,000,025  ;DELETE FILE
395             .WORD.  IO.RAT,000,026  ;READ FILE ATTRIBUTES
396             .WORD.  IO.WAT,000,027  ;WRITE FILE ATTRIBUTES
397             .WORD.  IO.APV,010,030  ;PRIVILEGED ACP CONTROL
398             .WORD.  IO.APC,000,030  ;ACP CONTROL
399     ;
400     ;
401             .MACRO  FILIO$  A
402             .ENDM   FILIO$
403             .ENDM   FILIO$
404
405     ;
406     ; DEFINE THE I/O FUNCTION CODES THAT ARE SPECIFIC TO INDIVIDUAL DEVICES
407     ;
408             .MACRO  SPCIO$  $$$GBL
409             .MCALL  .WORD.,DEFIN$
410             .IF     IDN,<$$$GBL>,<DEF$G>
411             ...GBL=1
412             .IFF
413             ...GBL=0
414             .ENDC
415     ;
416     ; I/O FUNCTION CODES FOR SPECIFIC DEVICE DEPENDENT FUNCTIONS
417     ;
418       .WORD.  IO.WLV,100,001  ;(DECTAPE) WRITE LOGICAL REVERSE
419       .WORD.  IO.WLS,010,001  ;(COMM.) WRITE PRECEDED BY SYNC TRAIN
420       .WORD.  IO.WNS,020,001  ;(COMM.) WRITE, NO SYNC TRAIN
421       .WORD.  IO.WAL,010,001  ;(TTY) WRITE PASSING ALL CHARACTERS
422       .WORD.  IO.WMS,020,001  ;(TTY) WRITE SUPPRESSIBLE MESSAGE
423       .WORD.  IO.CCO,040,001  ;(TTY) WRITE WITH CANCEL CONTROL-O
424       .WORD.  IO.WBT,100,001  ;(TTY) WRITE WITH BREAKTHROUGH
425       .WORD.  IO.WLT,010,001  ;(DISK) WRITE LAST TRACK
426       .WORD.  IO.WLC,020,001  ;(DISK) WRITE LOGICAL W/ WRITECHECK
427       .WORD.  IO.WPB,040,001  ;(DISK) WRITE PHYSICAL BLOCK
428       .WORD.  IO.WDD,140,001  ;(FLOPPY DISK) WRITE PHYSICAL W/ DELETED DATA
429       .WORD.  IO.RLV,100,002  ;(MAGTAPE,DECTAPE) READ REVERSE
430       .WORD.  IO.RST,001,002  ;(TTY) READ WITH SPECIAL TERMINATOR
431       .WORD.  IO.RAL,010,002  ;(TTY) READ PASSING ALL CHARACTERS
432       .WORD.  IO.RNE,020,002  ;(TTY) READ WITHOUT ECHO
433       .WORD.  IO.RNC,040,002  ;(TTY) READ - NO LOWER CASE CONVERT
```

```
434      .WORD.   IO.RTM,200,002  ;(TTY) READ WITH TIME OUT
435      .WORD.   IO.RDB,200,002  ;(CARD READER) READ BINARY MODE
436      .WORD.   IO.SCF,200,002  ;(DISK) SHADOW COPY FUNCTION
437      .WORD.   IO.RHD,010,002  ;(COMM.) READ, STRIP SYNC
438      .WORD.   IO.RNS,020,002  ;(COMM.) READ, DON'T STRIP SYNC
439      .WORD.   IO.CRC,040,002  ;(COMM.) READ, DON'T CLEAR CRC
440      .WORD.   IO.RPB,040,002  ;(DISK) READ PHYSICAL BLOCK
441      .WORD.   IO.RLC,020,002  ;(DISK,MAGTAPE) READ LOGICAL W/ READCHECK
442      .WORD.   IO.ATA,010,003  ;(TTY) ATTACH WITH AST'S
443      .WORD.   IO.GTS,000,005  ;(TTY) GET TERMINAL SUPPORT CHARACTERISTICS
444      .WORD.   IO.R1C,000,005  ;(AFC,AD01,UDC) READ SINGLE CHANNEL
445      .WORD.   IO.INL,000,005  ;(COMM.) INITIALIZATION FUNCTION
446      .WORD.   IO.TRM,010,005  ;(COMM.) TERMINATION FUNCTION
447      .WORD.   IO.RWD,000,005  ;(MAGTAPE,DECTAPE) REWIND
448      .WORD.   IO.SPB,020,005  ;(MAGTAPE) SPACE "N" BLOCKS
449      .WORD.   IO.SPF,040,005  ;(MAGTAPE) SPACE "N" EOF MARKS
450      .WORD.   IO.STC,100,005  ;SET CHARACTERISTIC
451      .WORD.   IO.SMD,110,005  ;(FLOPPY DISK) SET MEDIA DENSITY
452      .WORD.   IO.SEC,120,005  ;SENSE CHARACTERISTIC
453      .WORD.   IO.RWU,140,005  ;(MAGTAPE,DECTAPE) REWIND AND UNLOAD
454      .WORD.   IO.SMO,160,005  ;(MAGTAPE) MOUNT & SET CHARACTERISTICS
455      .WORD.   IO.HNG,000,006  ;(TTY) HANGUP DIAL-UP LINE
456      .WORD.   IO.RBC,000,006  ;READ MULTICHANNELS (BUFFER DEFINES CHANNELS)
457      .WORD.   IO.MOD,000,006  ;(COMM.) SETMODE FUNCTION FAMILY
458      .WORD.   IO.HDX,010,006  ;(COMM.) SET UNIT HALF DUPLEX
459      .WORD.   IO.FDX,020,006  ;(COMM.) SET UNIT FULL DUPLEX
460      .WORD.   IO.SYN,040,006  ;(COMM.) SPECIFY SYNC CHARACTER
461      .WORD.   IO.EOF,000,006  ;(MAGTAPE) WRITE EOF
462      .WORD.   IO.ERS,020,006  ;(MAGTAPE) ERASE TAPE
463      .WORD.   IO.DSE,040,006  ;(MAGTAPE) DATA SECURITY ERASE
464      .WORD.   IO.RTC,000,007  ;READ CHANNEL - TIME BASED
465      .WORD.   IO.SAO,000,010  ;(UDC) SINGLE CHANNEL ANALOG OUTPUT
466      .WORD.   IO.SSO,000,011  ;(UDC) SINGLE SHOT, SINGLE POINT
467      .WORD.   IO.RPR,000,011  ;(TTY) READ WITH PROMPT
468      .WORD.   IO.MSO,000,012  ;(UDC) SINGLE SHOT, MULTI-POINT
469      .WORD.   IO.RTT,001,012  ;(TTY) READ WITH TERMINATOR TABLE
470      .WORD.   IO.SLO,000,013  ;(UDC) LATCHING, SINGLE POINT
471      .WORD.   IO.MLO,000,014  ;(UDC) LATCHING, MULTI-POINT
472      .WORD.   IO.LED,000,024  ;(LPS11) WRITE LED DISPLAY LIGHTS
473      .WORD.   IO.SDO,000,025  ;(LPS11) WRITE DIGITAL OUTPUT REGISTER
474      .WORD.   IO.SDI,000,026  ;(LPS11) READ DIGITAL INPUT REGISTER
475      .WORD.   IO.SCS,000,026  ;(UDC) CONTACT SENSE, SINGLE POINT
476      .WORD.   IO.REL,000,027  ;(LPS11) WRITE RELAY
477      .WORD.   IO.MCS,000,027  ;(UDC) CONTACT SENSE, MULTI-POINT
478      .WORD.   IO.ADS,000,030  ;(LPS11) SYNCHRONOUS A/D SAMPLING
479      .WORD.   IO.CCI,000,030  ;(UDC) CONTACT INT - CONNECT
480      .WORD.   IO.LOD,000,030  ;(LPA11) LOAD MICROCODE
481      .WORD.   IO.MDI,000,031  ;(LPS11) SYNCHRONOUS DIGITAL INPUT
482      .WORD.   IO.DCI,000,031  ;(UDC) CONTACT INT - DISCONNECT
483      .WORD.   IO.XMT,000,031  ;(COMM.) TRANSMIT SPECIFIED BLOCK WITH ACK
484      .WORD.   IO.XNA,010,031  ;(COMM.) TRANSMIT WITHOUT ACK
485      .WORD.   IO.INI,000,031  ;(LPA11) INITIALIZE
486      .WORD.   IO.HIS,000,032  ;(LPS11) SYNCHRONOUS HISTOGRAM SAMPLING
487      .WORD.   IO.RCI,000,032  ;(UDC) CONTACT INT - READ
488      .WORD.   IO.RCV,000,032  ;(COMM.) RECEIVE DATA IN BUFFER SPECIFIED
489      .WORD.   IO.CLK,000,032  ;(LPA11) START CLOCK
490      .WORD.   IO.CSR,000,032  ;(BUS SWITCH) READ CSR REGISTER
491      .WORD.   IO.MDO,000,033  ;(LPS11) SYNCHRONOUS DIGITAL OUTPUT
492      .WORD.   IO.CTI,000,033  ;(UDC) TIMER - CONNECT
493      .WORD.   IO.CON,000,033  ;(COMM.) CONNECT FUNCTION
494                               ;(VT11) - CONNECT TASK TO DISPLAY PROCESSOR
495                               ;(BUS SWITCH) CONNECT TO SPECIFIED BUS
496      .WORD.   IO.STA,000,033  ;(LPA11) START DATA TRANSFER
```

```
497      .WORD.   IO.DTI,000,034  ;(UDC) TIMER - DISCONNECT
498      .WORD.   IO.DIS,000,034  ;(COMM.) DISCONNECT FUNCTION
499                               ;(VT11)-DISCONNECT TASK FROM DISPLAY PROCESSOR
500                               ;(BUS SWITCH) SWITCHED BUS DISCONNECT
501      .WORD.   IO.MDA,000,034  ;(LPS11) SYNCHRONOUS D/A OUTPUT
502      .WORD.   IO.DPT,010,034  ;(BUS SWITCH) DISCONNECT TO SPECIF PORT NO.
503      .WORD.   IO.RTI,000,035  ;(UDC) TIMER - READ
504      .WORD.   IO.CTL,000,035  ;(COMM.) NETWORK CONTROL FUNCTION
505      .WORD.   IO.STP,000,035  ;(LPS11,LPA11) STOP IN PROGRESS FUNCTION
506                               ;(VT11) - STOP DISPLAY PROCESSOR
507      .WORD.   IO.SWI,000,035  ;(BUS SWITCH) SWITCH BUSSES
508      .WORD.   IO.CNT,000,036  ;(VT11) - CONTINUE DISPLAY PROCESSOR
509      .WORD.   IO.ITI,000,036  ;(UDC) TIMER - INITIALIZE
510
511
512   ;
513   ; COMMUNICATIONS FUNCTIONS
514   ;
515
516      .WORD.   IO.CPR,010,033  ;CONNECT NO TIMEOUTS
517      .WORD.   IO.CAS,020,033  ;CONNECT WITH AST
518      .WORD.   IO.CRJ,040,033  ;CONNECT REJECT
519      .WORD.   IO.CBO,110,033  ;BOOT CONNECT
520      .WORD.   IO.CTR,210,033  ;TRANSPARENT CONNECT
521      .WORD.   IO.GNI,010,035  ;GET NODE INFORMATION
522      .WORD.   IO.GLI,020,035  ;GET LINK INFORMATION
523      .WORD.   IO.GLC,030,035  ;GET LINK INFO CLEAR COUNTERS
524      .WORD.   IO.GRI,040,035  ;GET REMOTE NODE INFORMATION
525      .WORD.   IO.GRC,050,035  ;GET REMOTE NODE ERROR COUNTS
526      .WORD.   IO.GRN,060,035  ;GET REMOTE NODE NAME
527      .WORD.   IO.CSM,070,035  ;CHANGE SOLO MODE
528      .WORD.   IO.CIN,100,035  ;CHANGE CONNECTION INHIBIT
529      .WORD.   IO.SPW,110,035  ;SPECIFY NETWORK PASSWORD
530      .WORD.   IO.CPW,120,035  ;CHECK NETWORK PASSWORD.
531      .WORD.   IO.NLB,130,035  ;NSP LOOPBACK
532      .WORD.   IO.DLB,140,035  ;DDCMP LOOPBACK
533
534   ;
535   ; ICS/ICR I/O FUNCTIONS
536   ;
537       .WORD.   IO.CTY,000,007  ;CONNECT TO TERMINAL INTERRUPTS
538       .WORD.   IO.DTY,000,015  ;DISCONNECT FROM TERMINAL INTERRUPTS
539       .WORD.   IO.LDI,000,016  ;LINK TO DIGITAL INTERRUPTS
540       .WORD.   IO.UDI,010,023  ;UNLINK FROM DIGITAL INTERRUPTS
541       .WORD.   IO.LTI,000,017  ;LINK TO COUNTER MODULE INTERRUPTS
542       .WORD.   IO.UTI,020,023  ;UNLINK FROM COUNTER MODULE INTERRUPTS
543       .WORD.   IO.LTY,000,020  ;LINK TO REMOTE TERMINAL INTERRUPTS
544       .WORD.   IO.UTY,030,023  ;UNLINK FROM REMOTE TERMINAL INTERRUPTS
545       .WORD.   IO.LKE,000,024  ;LINK TO ERROR INTERRUPTS
546       .WORD.   IO.UER,040,023  ;UNLINK FROM ERROR INTERRUPTS
547       .WORD.   IO.NLK,000,023  ;UNLINK FROM ALL INTERRUPTS
548       .WORD.   IO.ONL,000,037  ;UNIT ONLINE
549       .WORD.   IO.FLN,000,025  ;UNIT OFFLINE
550       .WORD.   IO.RAD,000,021  ;READ ACTIVATING DATA
551
552   ;
553   ; IP11 I/O FUNCTIONS
554   ;
555       .WORD.   IO.MAO,010,007  ;MULTIPLE ANALOG OUTPUTS
556       .WORD.   IO.LEI,010,017  ;LINK EVENT FLAGS TO INTERRUPT
557       .WORD.   IO.RDD,010,020  ;READ DIGITAL DATA
558       .WORD.   IO.RMT,020,020  ;READ MAPPING TABLE
559       .WORD.   IO.LSI,000,022  ;LINK TO DSI INTERRUPTS
```

```
560         .WORD.  IO.UEI,050,023  ;UNLINK EVENT FLAGS
561         .WORD.  IO.USI,060,023  ;UNLINK FROM DSI INTERRUPTS
562         .WORD.  IO.CSI,000,026  ;CONNECT TO DSI INTERRUPTS
563         .WORD.  IO.DSI,000,027  ;DISCONNECT FROM DSI INTERRUPTS
564
565     ;
566     ; PCL11 I/O FUNCTIONS
567     ;
568
569         .WORD.  IO.ATX,000,001  ;ATTEMPT TRANSMISSION
570         .WORD.  IO.ATF,000,002  ;ACCEPT TRANSFER
571         .WORD.  IO.CRX,000,031  ;CONNECT FOR RECEPTION
572         .WORD.  IO.DRX,000,032  ;DISCONNECT FROM RECEPTION
573         .WORD.  IO.RTF,000,033  ;REJECT TRANSFER
574
575
576         .MACRO  SPCIO$  A
577         .ENDM   SPCIO$
578         .ENDM   SPCIO$
579
580     ;
581     ; DEFINE THE I/O CODES FOR USER-MODE DIAGNOSITCS.  ALL DIAGNOSTIC
582     ; FUNCTION ARE IMPLEMENTED AS A SUBFUNCTION OF I/O CODE 10 (OCTAL).
583     ;
584
585         .MACRO  UMDIO$  $$$GBL
586         .MCALL  .WORD.,DEFIN$
587         .IF IDN <$$$GBL>,<DEF$G>
588     ...GBL=1
589             .IFF
590     ...GBL=0
591         .ENDC
592
593     ;
594     ; DEFINE THE GENERAL USER-MODE I/O QUALIFIER BIT.
595     ;
596
597         .WORD.  IQ.UMD,004,000  ;USER MODE DIAGNOSTIC REQUEST
598
599     ;
600     ; DEFINE USER-MODE DIAGNOSTIC FUNCTIONS.
601     ;
602
603         .WORD.  IO.HMS,000,010  ;(DISK) HOME SEEK OR RECALIBRATE
604         .WORD.  IO.BLS,010,010  ;(DISK) BLOCK SEEK
605         .WORD.  IO.OFF,020,010  ;(DISK) OFFSET POSITION
606         .WORD.  IO.RDH,030,010  ;(DISK) READ DISK HEADER
607         .WORD.  IO.WDH,040,010  ;(DISK) WRITE DISK HEADER
608         .WORD.  IO.WCK,050,010  ;(DISK) WRITECHECK (NON-TRANSFER)
609         .WORD.  IO.RNF,060,010  ;(DECTAPE) READ BLOCK NUMBER FORWARD
610         .WORD.  IO.RNR,070,010  ;(DECTAPE) READ BLOCK NUMBER REVERSE
611         .WORD.  IO.LPC,100,010  ;(MAGTAPE) READ LONGITUDINAL PARITY CHAR
612         .WORD.  IO.RTD,120,010  ;(DISK) READ TRACK DESCRIPTOR
613         .WORD.  IO.WTD,130,010  ;(DISK) WRITE TRACK DESCRIPTOR
614         .WORD.  IO.TDD,140,010  ;(DISK) WRITE TRACK DESCRIPTOR DISPLACED
615         .WORD.  IO.DGN,150,010  ;DIAGNOSE MICRO PROCESSOR FIRMWARE
616         .WORD.  IO.WPD,160,010  ;(DISK) WRITE PHYSICAL BLOCK
617         .WORD.  IO.RPD,170,010  ;(DISK) READ PHYSICAL BLOCK
618         .WORD.  IO.CER,200,010  ;(DISK) READ CE BLOCK
619         .WORD.  IO.CEW,210,010  ;(DISK) WRITE CE BLOCK
620
621     ;
622     ; MACRO REDEFINITION TO NULL
```

```
623     ;
624
625         .MACRO  UMDIO$  A
626         .ENDM
627
628
629         .ENDM   UMDIO$
630
631 ;
632 ; HANDLER ERROR CODES RETURNED IN I/O STATUS BLOCK ARE DEFINED THROUGH THIS
633 ; MACRO WHICH THEN CONDITIONALLY INVOKES THE MESSAGE GENERATING MACRO
634 ; FOR THE QIOSYM.MSG FILE
635 ;
636             .MACRO  .IOER.  SYM,LO,MSG
637             DEFIN$  SYM,LO
638             .IF     GT,$$MSG
639             .MCALL  .IOMG.
640             .IOMG.  SYM,LO,<MSG>
641             .ENDC
642             .ENDM   .IOER.
643     ;
644     ; I/O ERROR CODES ARE DEFINED THOUGH THIS MACRO WHICH THEN INVOKES THE
645     ; ERROR MESSAGE GENERATING MACRO, ERROR CODES -129 THROUGH -256
646     ; ARE USED IN THE QIOSYM.MSG FILE
647     ;
648             .MACRO  .QIOE.  SYM,LO,MSG
649             DEFIN$  SYM,LO
650             .IF     GT,$$MSG
651             .MCALL  .IOMG.
652             .IOMG.  SYM,<LO-128.>,<MSG>
653             .ENDC
654             .ENDM   .QIOE.
655     ;
656     ; CONDITIONALLY GENERATE DATA FOR WRITING A MESSAGE FILE
657     ;
658             .MACRO  .IOMG.  SYM,LO,MSG
659             .WORD   -^O<LO>
660             .ASCIZ  ^MSG^
661             .EVEN
662             .IIF    LT,^O<$$$MAX+<LO>>,$$$MAX=-^O<LO>
663             .ENDM   .IOMG.
664 ;
665 ; DEFINE SYMBOL SYM WHERE LO IS IS THE LOW ORDER BYTE, HI IS THE HIGH BYTE
666 ;
667             .MACRO  .WORD.  SYM,LO,HI
668             DEFIN$  SYM,<HI*400+LO>
669             .ENDM   .WORD.
```

Index

A

C

D

E

F

G

H

I

J

L

M

N

P

Q

R

S

T

U

V

Reader's Comments

This form is for document comments only. Digital will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of user/reader that you most nearly represent:

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify)_____

Name_____Date_____

Organization_____

Street_____

City_____State_____Zip Code_____
or Country

Do Not Tear - Fold Here and Tape

digital™

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO 33 MAYNARD MASS

POSTAGE WILL BE PAID BY ADDRESSEE

IAS Engineering/Documentation
Digital Equipment Corporation
5 Wentworth Drive GSF/L20
Hudson, NH 03051-4929

Do Not Tear - Fold Here